User's Guide – For Optimus Series

BASIC Compiler

Version 3.04

Printed on 20 March, 2006

Copyright Notice

Copyright © 2005 Metrologic Instruments Inc.

All rights reserved

The software contains proprietary information of Metrologic Instruments Inc.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

Due to continued product development this information may change without notice. The information and intellectual property contained herein is confidential between Metrologic and the client and remains the exclusive property of Metrologic. If you find any problems in the documentation, please report them to us in writing. The company does not warrant that this document is error-free.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Metrologic.

For product consultancy and technical support, please contact your local sales representative. Also, you may visit our web site for more information.

Contents

Copyright Notice II

Introduction 1

Typographical Conventions 2
Revision History 2

Development Environment 3

2.1 Disk Contents 3
2.2 System Requirements 4
2.3 BASIC Run-time Engine 5
2.4 Development Flow 5
2.4.1 Download Run-time Engine 6
2.4.2 Edit and Compile the BASIC Program 6
2.4.3 Download the BASIC Object Files 7

Using The BASIC Compiler 8

3.1 File Menu 8
3.2 Edit Menu 9
3.3 Configure Menu 11
3.4 Compile Menu 12
3.5 Help Menu 13

Basics of the BASIC Language 15

4.1 Constants 15
4.1.1 String 15
4.1.2 Numeric 15
4.2 Variables 16
4.2.1 Variable Names and Declaration Characters 16
4.2.2 Array Variables 16
4.3 Expression and Operators 17
4.3.1 Assignment Operator 17
4.3.2 Arithmetic Operator 18
4.3.3 Relational Operator 18
4.3.4 Logical Operator 18

4.4 Operator Precedence 19
4.5 Labels 19
4.6 Subroutines 20
4.7 Programming Style 21

BASIC Commands 22

5.1 General Commands 25
5.2 Commands for Decision Structures 28
5.3 Commands for Looping Structures 32
5.4 Commands for String Processing 34
5.4.1 Combining Strings 34
5.4.2 Comparing Strings 35
5.4.3 Getting the Length of a String 36
5.4.4 Searching for Strings 36
5.4.5 Retrieving Part of Strings 37
5.4.6 Converting for Strings 39
5.4.7 Creating Strings of Repeating Characters 42
5.5 Commands for Event Trapping 43
5.5.1 Event Triggers 43
5.5.2 Lock and Unlock 52
5.6 System Commands 54
5.7 Barcode Reader Commands 60
5.9 Keyboard Wedge Commands 69
5.9.1 Definition of the WedgeSetting Array 69
5.9.2 KBD / Terminal Type 70
5.9.3 Capital Lock Auto-Detection 71
5.9.4 Capital Lock Status Setting 71
5.9.5 Alphabets Case 71
5.9.6 Digits Position 72
5.9.7 Shift / Capital Lock Keyboard 72
5.9.8 Digit Transmission 72
5.9.9 Inter-Character Delay 72
5.9.10 Composition of Output String 73
5.10 Buzzer Commands 76
5.12 Calendar and Timer Commands 78
5.13 LED Command 81
5.14 Keypad Commands 82
5.15 LCD Commands 85
5.15.1 Graphic Display 86
5.15.2 Special Font Files 96
5.16 Battery Commands 96
5.17 Communication Ports 98
5.17.1 Port Mapping 98
5.17.2 RS-232, Serial IR and IrDA Communications 102
5.18 RF Communications 105
5.18.1 RF Models & Specifications 106
5.18.2 RF System Requirements 107
5.18.3 RF Terminal Properties 108
5.18.4 RF Base Properties 114
5.18.5 Programming RF Base - RS-232 Host Command Sets 115
5.19 Wireless Communications 121
5.19.1 Wireless Models 121
5.19.2 Network Configuration 121
5.19.3 IEEE 802.11b 127

5.19.4 Bluetooth 138
5.20 File Manipulation 147
5.20.1 DAT Files 148
5.20.2 DBF Files and IDX Files 154
5.20.3 Error Code 161
5.21 Memory 162
5.22 Debugging Commands 164
5.23 Reserved Host Commands 167

Barcode Setting **171**

Symbology Parameters 172
Code 39 172
Italian/French Pharmacode 172
Industrial/Interleave/Matrix 2 of 5 172
Codabar 173
UPCE 173
EAN8 173
EAN13 & UPCA 174
MSI 174
Plessey 175
Telepen 175
RSS14 175
RSS14 Limited 175
RSS14 Expanded 176
Scanner Parameters 176
Scan Mode 176
Read Redundancy 177
Time-Out 178
Negative Barcode 178

Run-Time Error Table **179**

Debugging Messages **181**

Index **191**

CHAPTER 1

Introduction

The BASIC Compiler provides users with a complete programming environment to develop application programs for the terminals in the BASIC language. The Windows-based Basic Compiler comes with a menu-driven interface to simplify software development and code modifications. Many system configurations, such as COM port properties and database file settings can be set up in the menus. Using this powerful programming tool to get rid of lengthy coding, users can develop an application for their own needs efficiently.

This manual is meant to provide detailed information about how to use the BASIC Compiler to write application programs for the portable terminals. It is organized in five chapters giving outlines as follows:

- Chapter 2 "Development Environment" - gives a concise introduction about the Basic Compiler, the development flow for applications, and the BASIC Compiler Run-time Engines.
- Chapter 3 "Using the BASIC Compiler" - gives a tour of the programming environment of the BASIC Compiler.
- Chapter 4 "Basics of the BASIC Language" - discusses the specific characteristics of the BASIC Language.
- Chapter 5 "BASIC Commands" - discusses all the supported BASIC functions and statements. More than 200 BASIC functions and statements are categorized according to their functions, and discussed in details.

The BASIC Compiler has been modified and improved since its first release in November 1997. Users can refer to RELEASE.TXT for detailed revision history.

In This Chapter

Typographical Conventions..... 2
Revision History 2

Typographical Conventions

Before you start using this guide, it is important to understand the terms and typographical conventions used in the documentation.

The following kinds of formatting in the text identify special information.

Formatting convention	Type of Information
Triangular Bullet (➢)	Highlight features or subjects that stand out from the text.
Special Bold	Items you must select, such as menu options, command buttons, or items in a list.
Emphasis	Use to emphasize the importance of a point or for variable expressions such as parameters.
CAPITALS	Names of keys on the keyboard/keypad. for example, SHIFT, CTRL, or ALT.
KEY+KEY	Key combinations for which the user must press and hold down one key and then press another, for example, CTRL+P, or ALT+F4.

Revision History

Version	Release Date	Notes
3.04	March 20, 2006	First Release.

CHAPTER 2

Development Environment

In This Chapter

2.1 Disk Contents 3
2.2 System Requirements 4
2.3 BASIC Run-time Engine 5
2.4 Development Flow..... 5

2.1 Disk Contents

The BASIC Compiler Kit contains two directories, namely, *BC* and *DOWNLOAD*. The purposes and contents of each directory are listed below.

To set up the BASIC programming environment on your PC, simply copy these two directories from the CD-ROM to your local hard disk.

- **BC: This directory contains the BASIC Compiler and the BASIC programs.**
  - Bc.exe: the BASIC Compiler.
  - Bc.chm: the on-line help file for the BASIC Compiler.
  - Synload.exe: the download program to download the Basic object files, .syn and .ini, to the terminals.
  - Release.txt: the revision history of the BASIC compiler.
  - Samples: including BASIC source files (.bas), initialization files (.ini) and BASIC object files (.syn) of the sample programs:

- **DOWNLOAD: This directory contains download utilities, the BASIC Run-time Engines and font files.**
  - Download.exe: the download utility to download the Motorola S format object file (.shx) to terminals via RS-232 or standard IrDA port.
  - IRLoad.exe: the download utility to download the Motorola S format object file (.shx) to terminals via Serial IR cradle.
  - BASIC Run-time Engines:

```
BAS_OPS.shx          // for Optimus S
BAS_OPR.shx          // for Optimus R
```

- Font files for Optimus S & R:

Font_OP_Japanese.shx	// Japanese	Font size: 16x16 (4 lines)
Font_OP_Japanese 12.shx	// Japanese	Font size: 12x12 (5 lines)
Font_OP_Korean.shx	// Korean	Font size: 16x16 (4 lines)
Font_OP_Korean12.shx	// Korean	Font size: 12x12 (5 lines)
Font_OP_Simplified_Chinese.shx	// Simplified Chinese	Font size: 16x16 (4 lines)
. Font_OP_Simplified_Chinese12.shx	// Simplified Chinese	Font size: 12x12 (5 lines)
Font_OP_Traditional_Chinese.shx	// Traditional Chinese	Font size: 16x16 (4 lines)
Font_OP_Simplified_Chinese 12.shx	// Traditional Chinese	Font size: 12x12 (5 lines)
Multi-Language-301.shx	// Multilanguage	

2.2 System Requirements

Before you install the BASIC Compiler, it is necessary to check that your PC meets the following minimum requirements:

Items	Requirements
CPU	Pentium 75MHz
Operating System	Windows 95/98/2000/NT/XP
Minimum RAM	16 MB
Minimum Hard Disk Space	20 MB

Note: Any terminal being programmed will need to have a minimum 128KB RAM.

2.3 BASIC Run-time Engine

The BASIC Run-time Engines work as interpreters of the BASIC commands. Terminals have to be loaded with the BASIC Run-time Engines to run the BASIC programs. Each model of terminal has its own Run-time Engine to drive its specific hardware features. The Run-time Engines are named as "BCxxx.shx", where "BCxxx" is the model number of the target terminal.

The BASIC Run-time also provides the capabilities for the user to configure the terminal. With the Run-time Engine loaded, the terminal can be set to the "System Mode". In the "System Mode", the user can set up the system settings such as the system clock and update the user program and so on. To enter the "System Mode", just press 7, 9 and POWER keys simultaneously. The System Menu presented in the "System Mode" varies for different models of terminals. For detailed functions of the System Menu, please refer to the user manual of the terminal.

2.4 Development Flow

Developing a BASIC program for terminals is as simple as counting 1-2-3. There are three steps:

Step 1 - download the BASIC Run-time to the target terminal.

Step 2 - edit and compile the BASIC program.

Step 3 - download the BASIC object file to the target terminal.

2.4.1 Download Run-time Engine

The BASIC Run-time Engines are programs being loaded on terminals to execute the BASIC object files. They must exist in the terminals before the BASIC object files are downloaded. To download the Run-time Engine (and/ or any other programs), the target terminal needs to be set to the "Download Mode" first to receive the new program. There are two ways to enter the "Download Mode": one is from the System Menu, and the other is from the Kernel Menu. For details of how to download a program, please refer to the terminal's user manual.

After the target terminal is set to the "Download Mode" and the connection to the host PC is properly established, the user can run one of the programs on the host PC to download the BASIC Run-time or any other *.shx* files to the terminal:

- ProgLoad.exe (for RS-232/IrDA, Cradle-IR or TCP/IP)

After the Run-time Engine is downloaded successfully, the message "Ready for BASIC Download" will be shown on the display of the terminal.

2.4.2 Edit and Compile the BASIC Program

The BASIC Compiler, *bc.exe*, comes with a text editor where users can edit their BASIC programs. Please refer to the next chapter for general information of the operation.

By default, the text being edited with the editor would be saved as a BASIC source file (*.bas*). The system settings defined in the configuration menus, including "Target Machine", COM port settings, transaction file settings, DBF settings and barcode settings would be saved as a system initialization file (*.ini*) with the same name when the *.bas* file is saved. The *.ini* file should be treated as part of the BASIC program, and should be included when the BASIC program is distributed.

If the BASIC program compiles without any errors, a BASIC object file (*.syn*) with the same name is generated. The *.ini* file and the *.syn* file are the two files to be downloaded to the terminal. The *.ini* file contains the system settings, while the *.syn* file contains the BASIC object code.

2.4.3 Download the BASIC Object Files

The user can use the BASIC Compiler or the standalone BASIC download utility, *Synload.exe*, to download a compiled BASIC program. *Synload.exe* provides only the download function of the BASIC Compiler, the user cannot use it to view or edit any BASIC code.

Both the *.ini* and *.syn* files must be downloaded to the target terminal. Be careful that if the *.ini* file is missing, the BASIC Compiler will download the default settings instead. In this case, it may cause errors during execution. In contrast to the BASIC Compiler, *Synload.exe* will not process the downloading if the *.ini* file is missing, and an error message will be shown on the display.

After the BASIC object file is downloaded, the target terminal will reboot itself to execute the BASIC program. If any run-time error occurs, an error message will be shown on the display. Please refer to Appendix II for a list of run-time errors. If the program is not running as desired, modify the BASIC source code and download it to the target terminal again.

CHAPTER 3

Using The BASIC Compiler

The BASIC Compiler looks like a traditional Windows environment application that supports file management, text editing, and some other functions to simplify the BASIC program development. There are five menus on the menu bar, and each menu provides several commands/items.

- File Menu
- Edit Menu
- Configure Menu
- Compile Menu
- Help Menu

This chapter discusses the function and operation of each command/item.

In This Chapter

3.1 File Menu..... 8
3.2 Edit Menu 9
3.3 Configure Menu..... 11
3.4 Compile Menu 12
3.5 Help Menu 13

3.1 File Menu

Six commands are provided on this menu.

Command	To Do...	
New	✓ Function	To create a new BASIC program.
	✓ Operation	Click "File" on the menu bar and select "New".
		For the same function, press hot key CTRL+ N or click the [New] icon on the tool bar.

Open	✓ Function	To open an existing BASIC program.
	✓ Operation	Click "File" on the menu bar and select "Open". For the same function, press hot key CTRL+ O or click the [Open] icon on the tool bar.
Save	✓ Function	To save the current editing BASIC program.
	✓ Operation	Click "File" on the menu bar and select "Save". For the same function, press hot key CTRL+ S or click the [Save] icon on the tool bar.
Save As	✓ Function	To save the current editing BASIC program with a new name.
	✓ Operation	Click "File" on the menu bar and select "Save As". Enter a new name in the pop-up window. Then click the [Save] button to save this program with the new file name.
Print	✓ Function	To print the current editing BASIC program.
	✓ Operation	Click "File" on the menu bar and select "Print". For the same function, press hot key CTRL+ P or click the [Print] icon on the tool bar.
Exit	✓ Function	To quit the BASIC Compiler.
	✓ Operation	Click "File" on the menu bar and select "Exit". For the same function, press hot key ALT+ F4.

3.2 Edit Menu

Seven commands are provided here to facilitate the editing of the BASIC source code.

Command	To Do...	
Undo	✓ Function	To abort the previous editing command or action.
	✓ Operation	Click "Edit" on the menu bar and select "Undo". For the same function, press hot key CTRL+ Z or click the [Undo] icon on the tool bar.

Cut

- ✓ Function — To cut a paragraph off the text and place it on the clipboard. The paragraph will be removed.
- ✓ Operation — Drag the cursor to select the paragraph to be cut off. This paragraph will be highlighted (in a reverse color). Click "Edit" on the menu bar and select "Cut".

  For the same function, press hot key CTRL+ X or click the [Cut] icon on the tool bar.

Copy

- ✓ Function — To copy a paragraph from the text to the clipboard.
- ✓ Operation — Drag the cursor to select the paragraph to be copied. This paragraph will be highlighted (in a reverse color). Click "Edit" on the menu bar and select "Copy".

  For the same function, press hot key CTRL+ C or click the [Copy] icon on the tool bar.

Paste

- ✓ Function — To paste a paragraph from the clipboard into the text. This paragraph will be inserted to the text.
- ✓ Operation — Move the cursor to the insertion point where the paragraph will be inserted, and left-click the mouse. Click "Edit" on the menu bar and select "Paste".

  For the same function, press hot key CTRL+ V or click the [Paste] icon on the tool bar.

Delete

- ✓ Function — To delete a paragraph from the text. This paragraph will not be placed on the clipboard.
- ✓ Operation — Drag the cursor to select the paragraph to be deleted. This paragraph will be highlighted (in a reverse color). Click "Edit" on the menu bar and select "Delete".

  For the same function, press the Del key.

Select All

- ✓ Function — To select all the contents of the text.
- ✓ Operation — Click "Edit" on the menu bar and select "Select All". All the contents will be highlighted (in a reverse color).

  For the same function, press hot key CTRL+ A.

Find

- ✓ Function — To find a specific letter, symbol, word, or paragraph in the text.

	✓ Operation	Click "Edit" on the menu bar and select "Find". In the pop-up window, enter the key word to be found in the text. Then, click the [Find] button to start searching. For the same function, press hot key CTRL+ F or click the [Find] icon on the tool bar.

3.3 Configure Menu

Seven items are provided here for the user to define the system settings.

The "Configure Transaction Files" and "Create DBF Files" provide the option of "Share file space with other applications". The 8xxx series terminals support multiple applications, but only one of them is active; this setting option allows different applications share the same files.

Command	To Do...	
Primary COM Port Setting	✓ Function	To set the properties of the primary COM port.
	✓ Operation	Click "Configure" on the menu bar and select "Primary COM Port Setting". Select the desired settings for each property in the pop-up window.

Secondary COM Port Setting	✓ Function	To set the properties of the secondary COM port.
	✓ Operation	Click "Configure" on the menu bar and select "Secondary COM Port Setting". Select the desired settings for each property in the pop-up window.
Configure Transaction Files	✓ Function	To define how many transaction files to be used and the data length for each transaction file.
	✓ Operation	Click "Configure" on the menu bar and select "Configure Transaction Files". In the pop-up window, check the box to enable the use of a transaction file, and type the data length for each enabled transaction file.
Create DBF Files	✓ Function	To define the DBF files to be used and the IDX files for each DBF file.
	✓ Operation	Click "Configure" on the menu bar and select "Create DBF Files". In the pop-up window, type the total record length for each DBF file and define the offset and length for the IDX files. (For the CPT-720 terminal, the user can choose to create DBF files on the Smart Media Card.)
Barcode Setting	✓ Function	To configure the system parameters for barcode symbologies and scanner performance.
	✓ Operation	Click "Configure" on the menu bar and select "Barcode Setting". In the pop-up window, check the box to enable the decodability of the target terminal to a particular barcode symbology. For the description of each barcode setting, please refer to Appendix I.

Note: When exiting the BASIC Compiler or opening another file, if the current file has not been changed but the barcode settings have been changed, the user will be asked whether to save the current file or not.

3.4 Compile Menu

Three commands are provided on this menu.

Command	To Do...	
Syntax checking	✓ Function	To check the syntax of the BASIC program.
	✓ Operation	Click "Compile" on the menu bar and select "Syntax checking". In the case of any syntax error in the BASIC program, the "Output" window pops up to show the line numbers and display the relevant syntax error message.
Compile	✓ Function	To compile the BASIC program.
	✓ Operation	Click "Compile" on the menu bar and select "Compile". For the same function, click the "Compile" icon on the tool bar. In the case of any syntax or compiling error, the "Output" window pops up to display the error messages. If the compilation is successfully done, the message "Build successfully, do you want to download the program?" will be shown on the screen. Click the [Yes] button if you want to download the program. (Refer to the "Download" command for downloading operation.)
Download	✓ Function	To download a compiled BASIC program to the target terminal.
	✓ Operation	Click "Compile" on the menu bar and select "Download". In the pop-up window, select the BASIC object file (.syn) to be downloaded, and then click [Open]. Select the correct COM port properties and then click [OK] to download. (For the CPT-7xx terminal, the user may select to download the BASIC program via Serial IR transceiver.) Please note the associated system initialization file (.ini) has to be in the same directory as the BASIC object file is; otherwise, the default system settings will be downloaded instead.

3.5 Help Menu

Three commands are provided on this menu.

Command	To Do...	
Contents	✓ Function	To display a table of contents for the BASIC online documentation.
	✓ Operation	Click "Help" on the menu bar and select "Contents". The online help file will be opened. Click on any topic for more detailed information.
Index	✓ Function	To display the help file index.
	✓ Operation	Click "Help" on the menu bar and select "Index". The online help file will be opened. Type a string for searching or directly click on the listed index entry for viewing further information.
About	✓ Function	To display the ownership and version of the program.
	✓ Operation	Click "Help" on the menu bar and select "About". The pop-up message box declares the ownership and version information of the program.

CHAPTER 4

Basics of the BASIC Language

In This Chapter

4.1 Constants 15
4.2 Variables 16
4.3 Expression and Operators 17
4.4 Operator Precedence 19
4.5 Labels 20
4.6 Subroutines 20
4.7 Programming Style 21

4.1 Constants

Constants are the actual values that BASIC uses during execution. There are two types of constants:

- String
- Numeric

4.1.1 String

A string constant is a sequence of up to 250 alphanumeric characters or symbols enclosed in a pair of double quotation marks.

- "Hello"
- "$20,000.00"
- "12 students"

4.1.2 Numeric

Numeric constants include positive and negative numbers. Numeric constants in BASIC cannot contain commas. There are three types of numeric constants that can be used in the BASIC Compiler:

- Integer Constants - Whole numbers between - 32,768 and + 32,767. No decimal point.
- Real Number Constants - Positive or negative real numbers, that is, numbers that contain a decimal point, such as 5.34 or - 10.0.
- Long Integer Constants - Whole numbers between - 2,147,483,648 and + 2,147,483,647.

4.2 Variables

Variables are symbols used to represent data items, such as numerical values or character strings that are used in a BASIC program. The value of a variable may be assigned explicitly and can be changed during the execution of a program. Be aware that the value of a variable is assumed to be undefined until a value is assigned to it.

4.2.1 Variable Names and Declaration Characters

The following are the rules for variable names and declaration characters:

- A variable name must begin with a letter (A to Z).
- The remaining characters can be letters, numbers, and/or underscores.
- The last character can be one of these type declaration characters:

`% integer`	: 2 bytes (- 32,768 to + 32,767)
`& long`	: 4 bytes (- 2,147,483,648 to + 2,147,483,647)
`! real number`	: 4 bytes
`$ string`	: 250 bytes
`nothing (default)`	: 2 bytes (- 32,768 to + 32,767)

- Only 4 types of variables are supported. The maximum number of variables is 1,000.
- The variable name cannot be a BASIC reserved word.
- Variable names are not case sensitive.

4.2.2 Array Variables

An array is a group or table of values referenced by the same variable name. Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression.

An array variable name has as many dimensions as there are subscripts in the array. For example,

`A(12)`	would reference a value in a one dimension array.
`T(2,5)`	would reference a value in a two-dimension array.
	... and so on.

- Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression. For example,

`DIM IntegerA%(20)`	: It declares an integer array with 20 elements.
`DIM StringB$(100)`	: It declares a string array with 100 elements.
`DIM RealC!(10)`	: It declares an integer array with 10 elements.
`DIM Tb(5,5)`	: It declares a two-dimension integer array with 5x5 elements.
`ArrayD(i+1,j)`	: The elements of an array are subscripted with an integer expression.

- The first element of an array is subscripted with 1.
- In the BASIC language, the maximum number of dimensions for an array is 2, and, up to 32,767 elements per dimension is allowed while compiling.

4.3 Expression and Operators

An expression may be a string or numeric constant, or a variable, or it may be a combination of constants and variables with operators to produce a single value.

Operators perform mathematical or logical operations. The operators provided by the BASIC Compiler may be divided into four categories, namely, *Assignment Operator*, *Arithmetic Operators*, *Relational Operators*, and *Logical Operators*.

4.3.1 Assignment Operator

The BASIC Compiler supports an assignment operator: "=".

For example,

- Length% = 100
- PI! = 3.14159
- Company$ = "Metrologic Instruments Inc."

4.3.2 Arithmetic Operator

The arithmetic operators are:

Operator	Operation	Sample Expression
^	Exponentiation	A% = 9^3
-	Negation (unary)	A% = -B%
*	Multiplication	A! = B! * C!
\	Division (integer)	A% = B! \ C!
/	Division (real)	A! = B! / C!
+	Addition	A% = B% + C%
-	Subtraction	A% = B% - C%
MOD	Modulo arithmetic	A% = B% MOD C%

4.3.3 Relational Operator

Relational operators are used to compare two values. The result of the comparison is either "True" or "False". This result may then be used to make a decision regarding program flow.

Operator	Operation	Sample Expression
=	Equality	A% = B%
<>	Inequality	A% <> B%
><	Inequality	A! >< B!
>	Greater than	A% > B!
<	Less than	A! < B!
>=	Greater than or equal to	A% >= B%
<=	Less than or equal to	A% <= B%

4.3.4 Logical Operator

Logical operators perform tests on multiple relations and Boolean operations. The logical operator returns a bit-wise result which is either "True" (not zero) or "False" (zero). In an expression, logical operations are performed after arithmetic and relational operations.

Operator	Operation	Sample Expression
NOT	Logical negation	IF NOT (A% = B%)
AND	Logical and	IF (A% = B%) AND (C% = D%)
OR	Inclusive or	IF (A% = B%) OR (C% = D%)

XOR	Exclusive or	IF (A% = B%) XOR (C% = D%)

4.4 Operator Precedence

The precedence of BASIC operators affects the evaluation of operands in expressions. Expressions with higher precedence operators are evaluated first. The precedence of BASIC operators is listed below in the order of precedence from highest to lowest. Where several operators appear together, they have equal precedence.

Order of Precedence	Type of Operation	Symbol
Highest	Arithmetic - Exponentiation	^
↓	Arithmetic - Multiplication, Division, Modulo	*, \, /, MOD
↓	Arithmetic - Addition, Subtraction	+, -
↓	Relational	=, <>, >, <, >=, <=
↓	Logical	AND, NOT, OR, XOR
Lowest	Assignment	=

4.5 Labels

Line labels are used to represent some special lines in the BASIC program. They can be either integer numbers or character strings.

- A valid integer number for the line label is in the range of 1 to 32,767.
- A character string label can have up to 49 characters. (If the string label has more than 49 characters, it will be truncated to 49 characters long.)
- The maximum number of labels is 1,000.
- The maximum lines can be compiled are 12,000.

A character string label that precedes a program line must have a colon ":" between the label and the program line, but it is not necessary for an integer label. For example,

```
        GOTO 100
        ...
100     PRINT "This is an integer label."
```

```
                ...
                GOTO Label2
                ...
Label2:         PRINT "This is a character string label."
```

4.6 Subroutines

A subroutine is a set of instructions given a particular name or a line label. Users can simplify their programming by breaking programs into smaller logical subroutines. A subroutine will be executed when being called by a *GOSUB* command. For example,

```
        ON KEY(1)GOSUB KeyF1
        ...
KeyF1:
        PRINT "F1 is pressed."
        RETURN
```

The command *RETURN* marks the end of the subroutine and tells the processor to return to the caller. A subroutine has to be appended at the end of the main BASIC program.

A subroutine can be defined with or without a pair of brackets. For example,

```
SUB Subroutine1( )
        ...
        PRINT "Subroutine1 is executed."
        END SUB
        ...
SUB Subroutine2
        ...
        PRINT "Subroutine2 is executed."
        END SUB
```

Since all the variables in the BASIC program are treated as global variables, passing arguments to subroutines is meaningless and enclosing arguments in the brackets of the subroutines will lead to a syntax error while compiling.

A subroutine in BASIC can be recursive, which means it can call itself or other subroutines that in turn call the first subroutine. The following sample program contains a recursive subroutine - Factorial, to calculate the value of n! ("n factorial").

```
      PRINT "Please enter a number (1 - 13):"
      INPUT N%
      FactResult! = 1
      Fact% = N%
      GOSUB Factorial
      PRINT N%, "! = ", FactResult
Loop:
      GOTO Loop
Factorial:
      IF Fact% < 1 THEN RETURN
      FactResult! = FactResult! * Fact%
      Fact% = Fact% -1
      GOSUB Factorial
      RETURN
```

4.7 Programming Style

The following are the guidelines used in writing programs in this manual, including the sample program. These guidelines are recommended for program readability, but they are not compulsory.

- Reserved words and symbolic constants appear in uppercase letters:

```
PRINT "Portable Terminal Demo Program"
BEEP(800,30,0,5,800,15,0,5,800,15)
```

- Variable names are in lowercase with an initial capital letter. If variable names are combined with more than one part, other capital letters may be used to make it easier to read:

```
ProcessFlag% = 0
Temp$ = GET_RECORD$(3,1)
```

- Line labels are used instead of line numbers:

```
ON READER(2) GOSUB GetSlotReader
```

CHAPTER 5

BASIC Commands

This chapter provides detailed descriptions of the commands supported by the BASIC Compiler. In addition to the commands commonly used in traditional versions of BASIC, a number of commands that deal with specific hardware features of the portable terminals are supported. These commands are within the user's BASIC programs to perform a wide variety of tasks, such as communications, LCD, buzzer, scanner, file manipulation, etc. They are categorized and described in this chapter by their functions or the resources they work on.

Some commands are suffixed with a dollar sign, $, which means a string is returned with the command. The compiler will accept these commands with or without the dollar sign. However, the dollar sign will be suffixed to these commands in this manual and the sample program.

The description for each BASIC command consists of five parts, *Purpose, Syntax, Remarks, Example and See Also*, which are further described below.

➢ Purpose

The purpose of the command is briefly explained.

➢ Syntax

According to the following conventions, the command syntax is described.

CAPS	BASIC keywords are indicated by capital letters.
Italics	Items in Italics represent variable information to be supplied by the user.
[ ]	Square brackets indicate optional parameters.
{ }	Braces indicate an item may be repeated as many times as necessary.
|	Vertical bar indicates alternative option.

➢ Remarks

Additional information regarding correct command usage is provided.

➢ Example

Various ways of using the statement are presented, including applicable and unusual modes of operation.

➢ See Also

List of related commands is provided, if there is any.

Note:	The types of terminals that support a specified BASIC command are listed to the right of the title bar of the command.

In This Chapter

5.1 General Commands 25
5.2 Commands for Decision Structures 28
5.3 Commands for Looping Structures..... 32
5.4 Commands for String Processing 34
5.5 Commands for Event Trapping 43
5.6 System Commands 54
5.7 General Reader Commands 60
5.8 RFID Reader Commands..... **Error! Bookmark not defined.**
5.9 Keyboard Wedge Commands 69
5.10 Buzzer Commands 76
5.11 Vibrator Commands 78
5.12 Calendar and Timer Commands 78
5.13 LED Command..... 81
5.14 Keypad Commands..... 82
5.15 LCD Commands 85
5.16 Battery Commands 96
5.17 Communication Ports 98
5.18 RF Communication..... 105
5.19 Wireless Communication..... 121
5.20 File Manipulation..... 147
5.21 Memory 162
5.22 Debugging Commands 164
5.23 Reserved Host Commands..... 167

5.1 General Commands

This section describes commands that are not confined to any specific hardware features.

ABS

Purpose To return the absolute value of a numeric expression.

Syntax *A* = ABS(*N*)

Remarks "*A*" is a numeric variable to be assigned to the absolute value of a numeric expression.

"*N*" is a numeric expression; it can be an integer or a real number.

Example TimeDifference% = ABS(Time1% - Time2%)

DIM

Purpose To specify the maximum value of variable subscripts and to allocate storage accordingly.

Syntax DIM *Array* (*range* {,*range*}) {, *Array*(*range* {,*range*})}

Remarks "*Array*" is an array variable.

"*range*" can be an integer or an integer expression.

The DIM statement sets all the elements of the specified arrays to an initial value of zero or empty string.

Note that the maximum allowable number of dimensions for an array is 2.

Example DIM A(10), B%(20), C$(30,10)

GOSUB

Purpose To call a specified subroutine.

Syntax GOSUB *SubName*\|*SubLabel*

Remarks "*SubName*" is the name of a subroutine.

"*SubLabel*" is the line label of a subroutine.

Example

```
GOSUB DoIt
...
  GOSUB Done
...
SUB DoIt( )
  PRINT "Now I've done it!"
END SUB
...
Done:
  PRINT "Now I've done it!"
RETURN
```

GOTO

Purpose To branch out unconditionally to a specified line number of line label from the normal program sequence.

Syntax GOTO *LineNumber*\|*LineLabel*

Remarks "*LineNumber*" is the integer number in front of a program line.

"*LineLabel*" is the string label of a program line.

Example

```
Loop:
  GOTO Loop
```

INT

Purpose To return the largest integer that is less than or equal to the given numeric expression.

Syntax *A%* = INT(*N*)

Remarks "*A%*" is an integer variable to be assigned to the result.

"*N*" is a numeric expression.

Example

```
A% = INT(-2.86)          ' A% = -3
B% = INT(2.86)           ' B% = 2
```

REM

Purpose To insert explanatory remarks in a program.

Syntax REM *remark*

' *remark*

Remarks "*remark*" may be any sequence of characters.

The BASIC compiler will ignore whatever follows the REM or ' until end of the line.

Example
```
REM This is a comment.          ' This is a comment.
```

SET_PRECISION

Purpose To set the precision of the decimal points for printing real number expressions.

Syntax SET_PRECISION(*N%*)

Remarks "*N%*" is a numeric expression in the range of 0 to 6.

The precision is set to two digits by default.

Example
```
PI! = 3.14159
PRINT "PI = ", PI!              ' result: PI = 3.14 (by default)
SET_PRECISION(6)
PRINT "PI = ", PI!              ' result: PI = 3.141590
SET_PRECISION(2)
PRINT "PI = ", PI!              ' result: PI = 3.14
```

SGN

Purpose To return an indication of the mathematical sign (+ or -) of a given numeric expression.

Syntax *A%* = SGN(*N*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
1	$N > 0$
0	$N = 0$
-1	$N < 0$

"*N*" is a numeric expression.

Example
```
A% = SGN(100)                   ' A% = 1
B% = SGN(-1.5)                  ' B% = -1
```

5.2 Commands for Decision Structures

Based on the value of an expression, decision structures cause a program to take one of the following two actions:

- To execute one of several alternative statements within the decision structure itself.
- To branch to another part of the program outside the decision structure.

In BASIC, decision-making is handled by the *IF...THEN...[ELSE...][ENDIF]* and *ON...GOSUB|GOTO...*statement.

The *IF...THEN...[ELSE...][ENDIF]* statement can be used anywhere the *ON...GOSUB|GOTO...* statement can be used. The major difference between the two statements is that *ON...GOSUB|GOTO...* evaluates a single expression, and then executes different statements or branches to different parts of the program based on the result. On the contrary, a block *IF...THEN...[ELSE...][ENDIF]* can evaluate completely different expressions. Moreover, the expression given in the ON *expression GOSUB|GOTO...* statement must be evaluated by a number in the range 1 to 255, while the expression in *IF...THEN...[ELSE...][ENDIF]* statement can only be evaluated as a TRUE or FALSE condition.

The *IF...THEN...[ELSE...][ENDIF]* statement can be nested up to 10 levels.

IF ... THEN ... [ELSE...]

Purpose To provide a decision structure for single-line conditional execution.

Syntax IF *condition* THEN *action1* [ELSE *action2*]

Remarks "*condition*" is a logical expression.

"*action*" is a BASIC statement.

Example IF Data1% > Data2% THEN Temp% = Data1% ELSE Temp% = Data2%

IF ... THEN ... {ELSE IF...} [ELSE...] END IF

Purpose To provide a decision structure for multiple-line conditional execution.

Syntax IF *condition1* THEN

Statementblock1

{ELSE IF *condition2* THEN

Statementblock2}

[ELSE

StatementblockN]

END IF

Remarks "*condition*" is a logical expression.

"*Statementblock*" can be multiple lines of BASIC statements.

Example

```
IF LEFT$(String1$,1) = "A" THEN
   PRINT "String1 is led by A."
ELSE IF LEFT$(String1$,1) = "B" THEN
   PRINT "String1 is led by B."
ELSE
   PRINT "String1 is not led by A nor B."
END IF
```

IF … THEN … END IF

Purpose To provide a decision structure for a conditional execution with multiple lines of actions.

Syntax

IF *condition1* THEN

action1

action2

…

END IF

Remarks "*condition*" is a logical expression.

"*action*" is a BASIC statement.

Example

```
IF Data1% > Large% THEN
   BEEP(800,30)
   Large% = Data1%
   PRINT "Current Largest Number is ", Data1%
END IF
```

ON ... GOSUB ...

Purpose To call one of the several specified subroutines depending on the value of the expression.

Syntax ON *N* GOSUB *SubName*\|*SubLabel* {, *SubName*\|*SubLabel*}

Remarks "*N*" is a numeric expression that is rounded to an integer. The value of *N* determines which subroutine is to be called. If the value of *N* is 0, or greater than the number of routines listed, the interpreter will continue with the next executable statement.

"*SubName*" is the name of a subroutine.

"*SubLabel*" is the line label of a subroutine.

Example

```
PRINT "Input a number (1-9):"
INPUT Num%
CLS
ON Num% GOSUB 100, 100, 100, 200, 200, 300, 400, 400, 400
...
100
   PRINT "Number 1-3 is input."
   RETURN
200
   PRINT "Number 4-5 is input."
   RETURN
300
   PRINT "6 is input."
   RETURN
400
   PRINT "Number 7-9 is input."
   RETURN
...
```

ON ... GOTO ...

Purpose To branch to one of several specified Line Labels depending on the value of an expression.

Syntax ON *N* GOTO *LineLabel* {, *LineLabel*}

Remarks "*N*" is a numeric expression which is rounded to an integer. The value of *N* determines which line label in the list will be used for branching. If the value *N* is 0, or greater than the number of line labels listed, the interpreter will continue with the next executable statement.

"*LineLabel*" is the string label of a program line.

Example

```
PRINT "Input a number (1-9):"
INPUT Num%
CLS
ON Num% GOTO 100, 100, 200, 200, 300, 400, 400, 400
...
100
   PRINT "Number 1-3 is input."
   GOTO 500
200
   PRINT "Number 4-5 is input."
   GOTO 500
300
   PRINT "6 is input."
   GOTO 500
400
   PRINT "Number 7-9 is input."
500
...
```

5.3 Commands for Looping Structures

Looping structures repeat a block of statements, either for a specified number of times or until a certain condition is matched. In BASIC, two kinds of looping structures, *FOR...NEXT* and *WHILE...WEND* can be used. Command *EXIT* can be used as an alternative to exit from both *FOR...NEXT* and *WHILE...WEND* loops.

Both *FOR...NEXT* and *WHILE...WEND* statements can be nested up to 10 levels.

EXIT

Purpose To provide an alternative exit for looping structures, such as FOR...NEXT and WHILE...WEND statements.

Syntax EXIT

Remarks EXIT can appear anywhere within the loop statement.

Example

```
DataCount% = TRANSACTION_COUNT
FOR Counter% = 1 TO DataCount%
   Data$ = GET_TRANSACTION_DATA$(Counter%)
   HostCommand$ = READ_COM$(1)
   IF HostCommand$ = "STOP" THEN EXIT
   WRITE_COM(1,Data$)
NEXT
```

FOR ... NEXT

Purpose To repeat the execution of a block of statements for a specified number of times.

Syntax

FOR *N%* = *startvalue* TO *endvalue* [STEP *step*]

[*Statement Block*]

NEXT [*N%*]

Remarks "*N%*" is an integer variable to be used as a loop counter.

"*startvalue*" is a numeric expression which is the initial value for the loop counter.

"*endvalue*" is a numeric expression which is the final value for the loop counter.

"*step*" is a numeric expression to be used as an increment/decrement of the loop counter The "step" is 1 by default.

If the loop counter ever reaches or beyond the endvalue, the program execution continues to the statement following the NEXT statement. The Statement block will be executed again otherwise.

Example

```
DataCount% = TRANSACTION_COUNT
FOR Counter% = 1 TO DataCount%
   Data$ = GET_TRANSACTION_DATA$(Counter%)
   WRITE_COM(1,Data$)
NEXT
```

WHILE ... WEND

Purpose To repeat the execution of a block of statements while a certain condition is TRUE.

Syntax

WHILE *condition*

[*Statement Block*]

WEND

Remarks If the "*condition*" is true, loop statements are executed until the WEND statement is encountered. Then the program execution returns to the WHILE statement and checks the condition again. If it is still true, the process will be repeated. Otherwise, the execution continues with the statement following the WEND statement.

Example

```
WHILE TRANSACTION_COUNT > 0
   Data$ = GET_TRANSACTION_DATA$(1)
   WRITE_COM(1,Data$)
   DEL_TRANSACTION_DATA(1)
WEND
```

5.4 Commands for String Processing

This section describes BASIC commands used to manipulate sequences of ASCII characters known as strings. In BASIC, strings are always variable length, from null to a maximum of 250.

5.4.1 Combining Strings

Two strings can be combined with the plus operator " +". The string following the plus operator is appended to the string preceding the plus operator. For example,

```
...
Data$ = DATE$ + TIME$ + EmployeeID$
SAVE_TRANSACTION(Data$)
...
```

5.4.2 Comparing Strings

Two strings can be compared with the relational operators, see section 4.3.3. A single character is greater than another character if its ASCII value is greater.

For example, the ASCII value of the letter "B" is greater than the ASCII value of the letter "A", so the expression "B" > "A" is true.

When comparing two strings, BASIC looks at the ASCII values of corresponding characters. The first character where the two strings differ determines the alphabetical order of the strings. For example, the strings "aaabaa" and "aaaaaaaa" are the same up to the fourth character in each, "b" and "a". Since the ASCII value of "b" is larger than that of "a", the expression "aaabaa" > "aaaaaaaa" is true.

If there is no difference between the corresponding characters of two strings and they are the same length, then the two strings are equal. If there is no difference between the corresponding characters of two strings, but one of the strings is longer, the longer string is greater than the shorter string. For example, "abc" = "abc" and "aaaaaaaa" > "aaaaa" are both true expressions.

Leading and trailing blank spaces are significant in comparing strings. For example, the string " abc" is less than the string "abc" since a blank space is less than an "a"; on the other hand, the string "abc " is greater than the string "abc".

5.4.3 Getting the Length of a String

LEN

Purpose To return the length of a string.

Syntax *A%* = LEN(*X$*)

Remarks "*A%*" is an integer variable to be assigned to the result.

"*X$*" may be a string variable, string expression, or string constant.

Note that non-printing characters and blanks are counted.

Example
```
String1$ = "abcde "
A% = LEN(String1$)                    'A% = 6, including the blank
```

5.4.4 Searching for Strings

Searching for a string inside another one is one of the most common string-processing tasks. *INSTR* is provided for this task.

INSTR

Purpose To search if one string exists inside another one.

Syntax *A%* = INSTR([*N%*,] *X$*, *Y$*)

Remarks "*A%*" is an integer variable to be assigned to the result.

"*N%*" is a numeric expression in the range of 1 to 255. Optional offset *N* sets the position for starting the search.

"*X$*", "*Y$*" may be a string variable, string expression, or string constant.

If *Y$* is found in *X$*, INSTR returns the position of the first occurrence of *Y$* in *X$*, from the starting point.

If *N* is larger than the length of *X$* or if *X$* is null, of if *Y$* cannot be found, INSTR returns 0.

If *Y$* is null, INSTR returns *N* (or 1 if *N* is not specified).

Example String1$ = "11025John Thomas, Accounting Manager"

String2$ = ","

EmployeeName$ = MID$(String1$, 6, INSTR(String1$, String2$) - 6)

' the employee's name starts at the sixth character

5.4.5 Retrieving Part of Strings

Several commands are provided to take strings apart by returning pieces of a string, from the left side, or the right side, or the middle of the target string.

LEFT$

Purpose To retrieve a given number of characters from the left side of the target string.

Syntax *A$* = LEFT$(*X$*, *N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is a numeric expression in the range of 0 to 255.

"*X$*" may be a string variable, string expression, or string constant.

If *N* is larger than the length of *X$*, the entire string (*X$*) is returned.

If *N* is zero, the null string (with lenght 0) is returned.

Example String1$ = "11025John Thomas, Accounting Manager"

EmployeeID$ = LEFT$(String1$, 5)

MID$

Purpose To retrieve a given number of characters from anywhere of the target string.

Syntax *A$* = MID$(*X$*, *N%*[, *M%*])

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" and "*M%*" are numeric expressions in the range of 0 to 255.

"*X$*" may be a string variable, string expression, or string constant.

MID$ function returns a string of length *M* characters from *X$* beginning with the *N*th character.

If *M* is omitted, or if there are fewer than *M* characters to the right of the *N*th character, all the characters beginning with the *N*th character to the rightmost are returned.

If *M* is equal to zero, or if *N* is greater than the length of *X$*, then MID$ returns a null string.

Example

```
String1$ = "11025John Thomas, Accounting Manager"
String2$ = ","
EmployeeName$ = MID$(String1$, 6, INSTR(String1$, String2$) - 6)
' the employee's name starts at the sixth character
```

RIGHT$

Purpose To retrieve a given number of characters from the right side of the target string.

Syntax *A$* = RIGHT$(*X$*, *N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is a numeric expression in the range of 0 to 255.

"*X$*" may be a string variable, string expression, or string constant.

If *N* is larger than the length of *X$*, the entire string is returned.

If *N* is zero, the null string (with length 0) is returned.

Example

```
String1$ = "11025John Thomas, Accounting Manager"
String2$ = ","
Title$ = RIGHT$(String1$, LEN(String1$) - INSTR(String1$, String2$))
```

TRIM_LEFT$

Purpose To return a copy of a string with leading blank spaces stripped away.

Syntax *A$* = TRIM_LEFT$(*X$*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*X$*" is a string variable that may contain some space characters at the beginning.

Example S1$ = TRIM_LEFT$(" Hello World!") ' S1$ = "Hello World!"

TRIM_RIGHT$

Purpose To return a copy of a string with trailing blank spaces stripped away.

Syntax *A$* = TRIM_RIGHT$(*X$*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*X$*" is a string variable that may contain some space characters at the end.

Example S2$ = TRIM_RIGHT$("Hello World! ") ' S2$ = "Hello World!"

5.4.6 Converting for Strings

Several commands are available for converting strings to uppercase or lowercase letters, as well as converting strings to numbers, and vice versa.

ASC

Purpose To return the decimal value for the ASCII code for the first character of a given string.

Syntax *A%* = ASC(*X$*)

Remarks "*A%*" is an integer variable to be assigned to the result.

"*X$*" is a string variable, consisting of characters.

Example A% = ASC("John Thomas") ' A% = 74

CHR$

Purpose To return the character for a given ASCII value.

Syntax *A%* = CHR$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is a numeric expression in the range of 0 to 255.

Example A$ = CHR$(65) ' A$ = "A"

HEX$

Purpose To return a string that represents the hexadecimal value (base 16) of the decimal argument.

Syntax *A$* = HEX$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is a numeric expression in the range of 0 to 2,147,483,647; it is rounded to an integer before HEX$(*N%*) is evaluated.

Example A$ = HEX$(140) ' A$ = "8C"

LCASE$

Purpose To return a copy of a string in which all uppercase letters will be converted to lowercase letters.

Syntax *A$* = LCASE$(*X$*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*X$*" may be a string variable, string expression, or string constant.

Example String1$ = "John Thomas"

String2$ = LCASE$(String1$) ' String2$ = "john thomas"

OCT$

Purpose To convert a decimal numeric expression to a string that represents the value of the numeric expression in octal notation.

Syntax *A$* = OCT$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is a numeric expression in the range 0 to 2,147,483,647; it is rounded to an integer before OCT$(*N%*) is evaluated.

Example A$ = OCT$(24) ' A$ = "30"

STR$

Purpose To convert a numeric expression to a string.

Syntax *A$* = STR$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is a numeric expression.

Example String$ = STR$(123)

UCASE$

Purpose To return a copy of a string in which all lowercase letters will be converted to uppercase letters.

Syntax *A$* = UCASE$(*X$*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*X$*" may be a string variable, string expression, or string constant.

Example
```
String1$ = "John Thomas"
String2$ = UCASE$(String1$)          ' String2$ = "JOHN THOMAS"
```

VAL

Purpose To return the numeric value of a string expression in long integer form.

Syntax *A&* = VAL$(*X$*)

Remarks "*A&*" is an integer/long integer variable to be assigned to the result.

"*X$*" is a string that includes numeric characters. If the first character is not numeric, VAL function returns 0. The VAL function will strip leading blanks, tabs, and linefeeds from the argument string. The return numeric value is in the range of -2,147,483,648 to 2,147,483,647.

Example
```
ON HOUR_SHARP GOSUB OnHourAlarm
...
OnHourAlarm:
  Hour% = VAL(LEFT$(TIME$,2))
  FOR Counter% = 1 TO Hour%
      BEEP(800,50)
      WAIT(200)
NEXT
RETURN
```

VALR

Purpose To convert a string expression to a real number.

Syntax *A!* = VALR(*X$*)

Remarks "*A!*" is a real number variable to be assigned to the result.

"*X$*" is a string that includes numeric characters. The precision of the converted result is governed by the "SET_PRECISION" function.

Example
```
A! = VALR("123.45")
PRINT "A = ", A!                    REM A = 123.45
...
```

5.4.7 Creating Strings of Repeating Characters

STRING$

Purpose To return a string containing the specified number of the requested character.

Syntax *A$* = STRING$(*N%, X$*)

*A$* = STRING$(*N%, J%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is a numeric expression in the range of 0 to 255, indicating the number of a character.

"*J%*" is a numeric expression in the range of 0 to 255, indicating the ASCII code of a character.

"*X$*" may be a string variable or string constant.

Example
```
IDX_LENGTH% = 20
Data$ = Name$ + STRING$(IDX_LENGTH% - LEN(Name$)," ")
ADD_RECORD$(1,Data$)
' padding with space if the length of Name$ is less than IDX_LENGTH%
```

5.5 Commands for Event Trapping

An event is an action recognized by the terminal, such as a function keystroke is detected (KEY event), a signal is received from the serial port (COM event), and so on. There are two ways to detect the occurrence of an event and reroute the program control to an appropriate subroutine: polling and trapping.

With event polling, the BASIC program explicitly checks for any event that happens at a particular point in its execution. For example, the following statements cause the program to loop back and forth until any key being pressed by the user:

```
Loop:
    KeyData$ = INKEY$
    IF KeyData$ = "" THEN GOTO Loop
   ...
```

Polling is useful when the occurrence of an event is predictable in the flow of the program. But if the time of the occurrence of an event is not predictable, trapping becomes the better alternative because the program will not be paused by the looping statements. For example, the following statements cause the program rerouting to the Key_F1 subroutine when the key F1 is pressed at anytime.

```
    ON KEY(1) GOSUB Key_F1
    ...
Key_F1:
    ...
```

This section describes a variety of events that the BASIC can trap as well as the associated commands.

5.5.1 Event Triggers

Below are 9 different events that can be trapped.

1. COM Event: a signal is received from the COM port.
2. ESC Event: the ESC key is pressed.
3. HOUR_SHARP Event: the system time is on the hour.
4. KEY Event: a function key is pressed.
5. MINUTE_SHARP Event: the system time is on the minute.

6 READER Event: a barcode data is decoded.

7 TCPIP Event: any data packet is received via TCP/IP.

8 TIMER Event: a time-out condition of an activated timer.

9 TOUCHSCREEN Event: a touchable item is activated by selecting.

OFF ALL

Purpose To disable all the event triggers.

Syntax OFF ALL

Remarks All the event triggers will be disabled. To resume the event trigger, the command ON *event* GOSUB... has to be called.

Example

```
ON READER(1) GOSUB BcrData_1
ON READER(2) GOSUB BcrData_2
ON KEY(1) GOSUB KeyData_1
...
IF BACKUP_BATTERY < BATTERY_LOW% THEN
   OFF ALL
   BEEP(2000,30)
   CLS
   PRINT "Backup Battery needs to be replaced!"
   Loop:
        GOTO Loop
END IF
…
```

See Also OFF COM, OFF ESC, OFF HOUR_SHARP, OFF KEY, OFF MINUTE_SHARP, OFF READER, OFF TCPIP, OFF TIMER, OFF TOUCHSCREEN

OFF COM

Purpose To terminate the "COM Event Trigger" that executes a specific subroutine when data is received from the COM ports.

Syntax OFF COM(*N%*)

Remarks "*N%*" is an integer variable, indicating the COM port.

COM	Model Number
1 ~ 2	7xx, 8000, 8300
1 ~ 4	8500

Example

```
ON COM(1) GOSUB HostCommand
...
HostCommand_1:
  OFF COM(1)    REM disable the trapping during data processing.
  ON COM(1) GOSUB HostCommand
  RETURN
```

See Also ON COM… GOSUB…

OFF ESC

Purpose To terminate the "ESC Event Trigger" that executes a specific subroutine when the ESC key is pressed.

Syntax OFF ESC

Remarks All the event triggers will be disabled. To resume the event trigger, the command ON *event* GOSUB... has to be called.

Example

```
ON ESC GOSUB Key_Esc
...
Key_Esc:
  OFF ESC
  …
  ON ESC GOSUB Key_Esc
  RETURN
```

See Also ON ESC GOSUB…

OFF HOUR_SHARP

Purpose To terminate the "HOUR_SHARP Event Trigger" that executes a specific subroutine when the system time is on the hour.

Syntax OFF HOUR_SHARP

Remarks

Example OFF HOUR_SHARP

See Also ON HOUR_SHARP GOSUB…

OFF KEY

Purpose To terminate the "FUNCTION KEY Event Trigger" that executes a specific subroutine when a function key is pressed.

Syntax OFF KEY(*number%*)

Remarks "*number%*" is an integer variable in the range of 1 to 12, indicating a function key of the keypad.

Example

```
ON KEY(1) GOSUB On_Shift
ON KEY(2) GOSUB Off_Shift
...
On_Shift:
  OFF KEY
  Mode$ = "IN"
  GOSUB Process
  ON KEY(1) GOSUB On_Shift
  RETURN
  ...
```

See Also ON KEY… GOSUB…

OFF MINUTE_SHARP

Purpose To terminate the "MINUTE_SHARP Event Trigger" that executes a specific subroutine when the system time is on the minute.

Syntax OFF MINUTE_SHARP

Remarks

Example OFF MINUTE_SHARP

See Also ON MINUTE_SHARP GOSUB…

OFF READER

Purpose To terminate the "READER Event Trigger" that executes a specific subroutine when data is received from the reader ports.

Syntax OFF READER(*N%*)

Remarks "*N%*" is an integer variable, indicating the reader port.

It is 1 for portable terminals.

Example

```
ON READER(1) GOSUB BcrData_1
...
BcrData_1:
  OFF READER(1)
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  CLS
  PRINT Data$
```

See Also ON READER… GOSUB…

OFF TCPIP 8xxx

Purpose To terminate the "TCP/IP Event Trigger".

Syntax OFF TCPIP

Remarks "*SubLabel*" is the line label of a subroutine.

Example OFF TCPIP

See Also ON TCPIP GOSUB…

OFF TIMER

Purpose To terminate the "TIMER Event Trigger" that is specified by the ON TIMER... GOSUB... command.

Syntax OFF TIMER(*N%*)

Remarks "*N%*" is an integer variable in the range of 1 to 5, indicating the number of the timer specified by the user.

Example

```
ON TIMER(1,200) GOSUB ClearScreen          ' TIMER(1) = 2 sec
  ...
ClearScreen:
  OFF TIMER(1)
  CLS
  RETURN
```

See Also ON TIMER… GOSUB…

ON COM ... GOSUB ...

Purpose To activate the "COM Event Trigger" that executes a specific subroutine when data is received from the COM ports.

Syntax ON COM(*N%*) GOSUB *SubName|SubLabel*

Remarks "*N%*" is an integer variable, indicating the COM port.

COM	Model Number
1 ~ 2	7xx, 8000, 8300
1 ~ 4	8500

Example

```
ON COM(1) GOSUB HostCommand
  ...
HostCommand_1:
  OFF COM(1)
  ...
  ON COM(1) GOSUB HostCommand
  RETURN
```

See Also Communication Ports commands, OFF COM

ON ESC GOSUB ...

Purpose To activate the "ESC Event Trigger" that executes a specific subroutine when the ESC key is pressed.

Syntax ON ESC GOSUB *SubName|SubLabel*

Remarks "*SubName|SubLabel*" is the name or line label of a subroutine to be called when the event is triggered.

Example

```
ON ESC GOSUB Key_Esc
...
Key_Esc:
  OFF ESC
  ...
  ON ESC GOSUB Key_Esc
    RETURN
```

See Also Keypad commands, OFF ESC

ON HOUR_SHARP GOSUB ...

Purpose To activate the "HOUR_SHARP Event Trigger" that executes a specific subroutine when the system time is on the hour.

Syntax ON HOUR_SHARP GOSUB *SubName|SubLabel*

Remarks "*SubName|SubLabel*" is the name or line label of a subroutine to be called when the event is triggered.

Example

```
...
ON HOUR_SHARP GOSUB OnHourAlarm
...
OnHourAlarm:
  CurrentTime$ = TIME$
  Hour% = VAL(LEFT$(CurrentTime$,2))
  FOR I = 1 TO Hour%
    BEEP(800,10,0,10)
    WAIT(100)
  NEXT
  RETURN
```

See Also Calendar & Timer commands, OFF HOUR_SHARP

ON KEY ... GOSUB ...

Purpose To activate the "FUNCTION KEY Event Trigger" that executes a specific subroutine when a function key is pressed.

Syntax ON KEY(*number%*) GOSUB *SubName|SubLabel*

Remarks "*number%*" is an integer variable in the range of 1 to 12 (1 to 9 for 711), indicating the function key of the keypad.

"*SubName**SubLabel*" is the name or line label of a subroutine to be called when the event is triggered.

Example

```
ON KEY(1) GOSUB On_Shift
ON KEY(2) GOSUB Off_Shift
...
On_Shift:
  Mode$ = "IN"
    RETURN
Off_Shift:
  Mode$ = "OUT"
    RETURN
```

See Also Keypad commands, OFF KEY

ON MINUTE_SHARP GOSUB ...

Purpose To activate the "MINUTE_SHARP Event Trigger" that executes a specific subroutine when the system time is on the minute.

Syntax ON MINUTE_SHARP GOSUB *SubName**SubLabel*

Remarks "*SubName**SubLabel*" is the name or line label of a subroutine to be called when the event is triggered.

Example

```
...
ON MINUTE_SHARP GOSUB CheckTime
...
CheckTime:
  CurrentTime$ = TIME$
  Hour% = VAL(MID$(CurrentTime$,3,2))
  IF Hour% = 30 THEN GOSUB HalfHourAlarm
  RETURN
  ...
HalfHourAlarm:
  BEEP(800,30)
  WAIT(100)
  RETURN
```

See Also Calendar & Timer commands, OFF MINUTE_SHARP

ON READER ... GOSUB ...

Purpose To activate the "READER Event Trigger" that executes a specific subroutine when data is received from the reader ports.

Syntax ON READER(*N%*) GOSUB *SubName**SubLabel*

Remarks "*N%*" is an integer variable, indicating the reader port.

It is 1 for portable terminals.

"*SubName**SubLabel*" is the name or line label of a subroutine to be called when the event is triggered.

Example

```
ON READER(1) GOSUB BcrData_1
...
BcrData_1:
  OFF READER(1)
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  ...
```

See Also General Reader commands, OFF READER

ON TCPIP GOSUB... WLAN

Purpose To activate the "TCP/IP Event Trigger" which executes a specific subroutine when data is received from any of the TCP/IP connections or some error is taking place.

Syntax ON TCPIP GOSUB *SubLabel*

Remarks "*SubLabel*" is the line label of a subroutine.

The GET_TCPIP_MESSAGE routine is used to identify the status of TCP/IP connections.

Example

```
ON TCPIP GOSUB TCPIP_Trigger
...
TCPIP_Trigger:
MSG = GET_TCPIP_MESSAGE
...
```

See Also OFF TCPIP, TCP/IP Networking commands

ON TIMER ... GOSUB ...

Purpose To activate the "TIMER Event Trigger" that executes a specific subroutine when the system runs out of the time duration specified by the user.

Syntax ON TIMER(*N%, duration%*) GOSUB *SubName\SubLabel*

Remarks "*N%*" is an integer variable in the range of 1 to 5, indicating the ordinal number of timer.

"*duration%*" is an integer variable, indicating a specified period of time in units of 10 ms.

"*SubName\SubLabel*" is the name or line label of a subroutine to be called when the event is triggered.

Up to five timers can be set in a BASIC program. Be sure the timer IDs are different. Otherwise, the latter created timer will overwrite the former one.

Example

```
ON TIMER(1, 200) GOSUB ClearScreen          ' TIMER(1) = 2 sec
...
ClearScreen:
  OFF TIMER(1)
  CLS
  RETURN
```

See Also Calendar & Timer commands, OFF TIMER

5.5.2 Lock and Unlock

Event trapping could be nested. If the event triggers are activated in a BASIC program, it is possible that an event-driven subroutine can be interrupted by any upcoming events. Normally, the new event would be processed first. In some cases where we don't want the event-driven subroutine to be interrupted by other events, the *LOCK* and *UNLOCK* commands can be used to hold off new events.

LOCK

Purpose To hold all the activated event triggers until they are released by the UNLOCK command.

Syntax LOCK

Remarks Command LOCK can prevent the nesting of event triggers. All the activated event triggers will be disabled until UNLOCK is called.

Example

```
ON READER(1) GOSUB BcrData_1
ON READER(2) GOSUB BcrData_2
...
BcrData_1:
  LOCK
  BEEP(2000,5)
  Data$ = GET_READER_DATA(1)$
  GOSUB AddNewData
  UNLOCK
  RETURN

BcrData_2:
  BEEP(2000,5)
  Data$ = GET_READER_DATA(2)$
  GOSUB AddNewData
  RETURN
```

In this example, the BASIC program can trap the READER(1) and READER(2) events and reroute to the subroutines BcrData_1 and BcrData_2 respectively. In BcrData_1, the command LOCK disables all the activated event triggers so that the subroutine BcrData_1 will not be interrupted by a new upcoming READER(1) and/or READER(2) event. On the other hand, since LOCK is not called in BcrData_2, any new coming READER(1) and READER(2) event will interrupt the ongoing BcrData_2, and therefore, may affect the expected results.

See Also UNLOCK

UNLOCK

Purpose To release all the activated event triggers held by the LOCK command.

Syntax UNLOCK

Remarks

Example Refer to the LOCK command.

See Also LOCK

5.6 System Commands

This section describes the system commands, such as the commands to change the CPU running speed, get the device ID, and/or restart the system.

AUTO_OFF

Purpose To set a specified period of time for the system to automatically shut down the user program as long as there is no operation in the interval.

If the time interval is set to zero, this function will be disabled.

Syntax AUTO_OFF(*N%*)

Remarks "*N%*" is an integer variable, indicating a specified period of time in units of 1 second.

Example
```
AUTO_OFF(30)        ' auto off after 30 seconds
AUTO_OFF(0)         ' disable the AUTO OFF function
```

See Also POWER_ON, RESTART

CHANGE_SPEED

Purpose To change the CPU running speed.

Syntax CHANGE_SPEED(*N%*)

Remarks "*N%*" is an integer variable in the range of 1 to 5, indicating the CPU running speed.

Value	Meaning
1	Sixteenth speed
2	Eighth speed
3	Quarter speed
4	Half speed
5	Full speed

When the system is not heavy loaded, i.e. waiting for data input, it is suggested to change the CPU running speed to a lower level to reduce the power consumption.

Example
```
CHANGE_SPEED(3)
```

DEVICE_ID$

Purpose To get the serial number of the terminal.

Syntax *A$* = DEVICE_ID$

Remarks "*A$*" is a string variable to be assigned to the result.

A string of the terminal's serial number will be returned.

Example PRINT "S/N:", DEVICE_ID$

GET_LANGUAGE Optimus

Purpose To retrieve the font/language setting.

Syntax *A%* = GET_LANGUAGE

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning	Font Size or Code Page
0	System font	
1	Traditional Chinese font	Big-5, 16x16 dots
2	Simplified Chinese font	GB code, 12x16 dots
3	Simplified Chinese font	GB code, 16x16 dots
4	Korean font	
5	Japanese font	
6	Hebrew font	
7	Polish font	
8	Russian font	
9	Traditional Chinese font	Big-5, 12x12 dots
10	Thai font	12x12 dots
11	Simplified Chinese font	GB code, 12x12 dots
12	Japanese font	12x12 dots
16	English	MS-DOS Code page 437
17	Canadian French	MS-DOS Code page 863
18	Hebrew	MS-DOS Code page 862
19	Multilingual Latin I	MS-DOS Code page 850
20	Nordic	MS-DOS Code page 865
21	Portuguese	MS-DOS Code page 860
22	Cyrillic	Windows Code page 1251
23	Latin II	MS-DOS Code page 852
24	Central European, Latin II	Windows Code page 1250
25	Turkish	MS-DOS Code page 857
26	Latin II	
27	Windows 1250	
28	ISO-28592 (Latin 2)	ISO 8859-2
29	IBM-LATIN II	
30	Greek	MS-DOS Code page 737
31	Latin I	Windows Code page 1252
32	Greek	Windows Code page 1253

Example language% = GET_LANGUAGE

See Also SET_LANGUAGE

MENU

Purpose To create a menu and let the user to select an item by using (1) the shortcut keys or (2) the UP/DOWN arrow keys, and then the ENTER key to confirm the selection. Also, allow the use of the ESC key to cancel the current operation.

Syntax *A%* = MENU(Item$)

Remarks "*A%*" is an integer variable to be assigned to the result; it is the ordinal number of the menu item that the user has selected.

"*Item$*" is a string variable, indicating the menu items that are separated and ended by carriage return (CR, 0x0d).

Note that the following features are applied to the 8xxx series only -

- Shortcut key: & (It is restricted to only one character next to &.)
- Menu title: @ (The title can be put anywhere in the menu string.)
- Display the Up/Down arrow icons

Example Below is an illustrative example:

```
MENU_STR$ = "1 INFORMATION" + CHR$(13)
MENU_STR$ = MENU_STR$ + "@SYSTEM MENU" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&2 SETTINGS" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&3 TESTS" + CHR$(13)
MENU_STR$ = MENU_STR$ + "4 LOAD PROGRAM" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&5 BLUETOOTH MENU" + CHR$(13)
...
S% = MENU(MENU_STR$)
...
```

POWER_ON

Purpose To determine whether to restart or resume the program upon powering on.

Syntax POWER_ON(*N%*)

Remarks "*N%*" can be 0 or 1.

Value	Meaning
0	Resume
1	Restart

Example

```
POWER_ON(0)                    ' set to resume mode
```

See Also AUTO_OFF, RESTART

RESTART

Purpose To restart the system.

Syntax RESTART

Remarks The RESTART command will terminate the execution of the BASIC program and restart it.

Example

```
HostCommand$ = READ_COM$(1)
...
IF HostCommand$ = "RESTART" THEN
   RESTART
ELSE
...
```

See Also AUTO_OFF, POWER_ON

SET_LANGUAGE Optimus

Purpose To make use of a font (language) by selecting it from the multi-language font file.

Syntax SET_LANGUAGE(*N*%)

Remarks Note that this function will fail if the multi-language font file does not exist.

"*N%*" is an integer variable in the range of 16 to 32.

Value	Meaning	Font Size or Code Page
16	English	MS-DOS Code page 437
17	Canadian French	MS-DOS Code page 863
18	Hebrew	MS-DOS Code page 862
19	Multilingual Latin I	MS-DOS Code page 850
20	Nordic	MS-DOS Code page 865
21	Portuguese	MS-DOS Code page 860
22	Cyrillic	Windows Code page 1251
23	Latin II	MS-DOS Code page 852
24	Central European, Latin II	Windows Code page 1250
25	Turkish	MS-DOS Code page 857
26	Latin II	
27	Windows 1250	
28	ISO-28592 (Latin 2)	ISO 8859-2
29	IBM-LATIN II	
30	Greek	MS-DOS Code page 737
31	Latin I	Windows Code page 1252
32	Greek	Windows Code page 1253

Example SET_LANGUAGE (17) ' select French

See Also GET_LANGUAGE

SYSTEM_PASSWORD

Purpose To set the password protection for entering the system menu.

Syntax SYSTEM_PASSWORD(*A$*)

Remarks "*A$*" is a string constant or variable, representing the password.

Example SYSTEM_PASSWORD("12345")

VERSION

Purpose To write version information to the system.

Syntax VERSION(*A$*)

Remarks "*A$*" is a string variable, indicating program name, date, etc.

The user can use this command to write version information to the system. Such information can be checked from the "Version" submenu of the "System Menu".

Note that this command must be on the first line of the program; otherwise, it will be ignored. The string for version information cannot exceed 15 characters.

Example VERSION("CipherBASIC 2.0")

...

5.7 Barcode Reader Commands

The terminals are able to read barcode data from the reader ports. This section describes the BASIC commands that are related to the reader ports of the terminals.

Commands for triggering the READER event: *OFF READER, ON READER GOSUB...*

CODE_TYPE

Purpose To get the type of symbology being decoded upon a successful scan.

Syntax *A%* = CODE_TYPE

Remarks "*A%*" is an integer variable to be assigned to the result.

The following list shows the possible values of CODE_TYPE.

Symbology	Type	Symbology	Type
Code 39	65(A)	EAN8	77(M)
Italian Pharmacode	66(B)	EAN8 with Addon 2	78(N)
CIP 39	67(C)	EAN8 with Addon 5	79(O)
Industrial 25	68(D)	EAN13	80(P)
Interleave 25	69(E)	EAN13 with Addon 2	81(Q)
Matrix 25	70(F)	EAN13 with Addon 5	82(R)
Codabar (NW7)	71(G)	MSI	83(S)
Code 93	72(H)	Plessey	84(T)
Code 128	73(I)	EAN 128	85(U)
UPCE	74(J)		
UPCE with Addon 2	75(K)	Telepen	90 (Z)
UPCE with Addon5	76(L)	RSS14	91 ([)

Example ...

```
CheckCodeType:
   IF CODE_TYPE = 65 THEN
      BcrType$ = "Code 39"
   ELSE IF CODE_TYPE = 66 THEN
      BcrType$ = "Italian Pharmacode"
...
END IF
PRINT "Code Type:", BcrType$
RETURN
```

See Also GET_READER_SETTING, READER_SETTING

DISABLE READER

Purpose To disable the reader ports of the terminal.

Syntax DISABLE READER(*N%*)

Remarks "*N%*" is an integer variable, indicating the reader port; it is 1 for portable terminals.

Example DISABLE READER(1)

See Also ENABLE READER, GET_READER_DATA$

ENABLE READER

Purpose To enable the reader ports of the terminal.

Syntax ENABLE READER(*N%*)

Remarks "*N%*" is an integer variable, indicating the reader port; it is 1 for portable terminals.

The reader ports are disabled by default. To enable barcode decoding function, the reader ports have to be enabled by the ENABLE READER command.

Example

```
ENABLE READER(1)
ON READER(1) GOSUB Bcr_1
...
Bcr_1:
Data$ = GET_READER_DATA$(1)
RETURN
```

See Also DISABLE READER, GET_READER_DATA$, OFF READER, ON READER GOSUB…

GET_READER_DATA$

Purpose To get data that is read from a specified reader port.

Syntax *A$* = GET_READER_DATA$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is an integer variable, indicating the reader port; it is 1 for portable terminals.

Usually, the user uses ON READER GOSUB command to trap the event the data is transmitted to the terminal through the reader port, and then uses GET_READER_DATA$ command in a subroutine to get the reader data.

Example

```
ENABLE READER(1)
ON READER(1) GOSUB Bcr_1
...
Bcr_1:
Data$ = GET_READER_DATA$(1)
RETURN
```

See Also DISABLE READER, ENABLE_READER, OFF READER, ON READER GOSUB…

GET_READER_SETTING

Purpose To get the value of a specified parameter of the barcode settings.

Syntax *A%* = GET_READER_SETTING(*N%*)

Remarks "*A%*" is a string variable to be assigned to the result.

"*N%*" is an integer variable, indicating the number of a parameter. (cf. READER_SETTING)

Example

```
Setting1% = GET_READER_SETTING(1)
IF Setting1% = 1 THEN
   PRINT "Code 39 readability is enabled."
ELSE
   PRINT "Code 39 readability is disabled."
END IF
```

See Also CODE_TYPE, READER_SETTING

READER_SETTING

Purpose To set the value of a specified parameter of the barcode settings.

Syntax READER_SETTING(*N1%, N2%*)

Remarks "*N1%*" is an integer variable, indicating the number of a parameter.

"*N2%*" is an integer variable, indicating the value to be set to a parameter.

A set of parameters called barcode settings determines how the decoder will decode the barcode data. The initial values of the barcode settings are given by the Barcode Settings Window of the BASIC Compiler. Refer to Appendix I for details of the settings. The user can reset the values by using the READER_SETTING command in a BASIC program. The following table shows the details of these parameters.

No.	Values and Description
1	1: Enable Code 39 0: Disable Code 39
2	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode

3	1: Enable CIP 39 0: Disable CIP 39
4	1: Enable Industrial 25 0: Disable Industrial 25
5	1: Enable Interleave 25 0: Disable Interleave 25
6	1: Enable Matrix 25 0: Disable Matrix 25
7	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)
8	1: Enable Code 93 0: Disable Code 93
9	1: Enable Code 128 0: Disable Code 128
10	1: Enable UPCE 0: Disable UPCE
11	1: Enable UPCE Addon 2 0: Disable UPCE Addon 2
12	1: Enable UPCE Addon 5 0: Disable UPCE Addon 5
13	1: Enable EAN8 0: Disable EAN8
14	1: Enable EAN8 Addon 2 0: Disable EAN8 Addon 2
15	1: Enable EAN8 Addon 5 0: Disable EAN8 Addon 5
16	1: Enable EAN13 0: Disable EAN13
17	1: Enable EAN13 Addon 2 0: Disable EAN13 Addon 2
18	1: Enable EAN13 Addon 5 0: Disable EAN13 Addon 5
19	1: Enable MSI 0: Disable MSI
20	1: Enable Plessey (to be continued)

	0: Disable Plessey
21	Reserved
22	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character
23	1: Verify Code 39 Check Character 0: DO NOT verify Code 39 Check Character
24	1: Transmit Code 39 Check Character 0: DO NOT transmit Code 39 Check Character
25	1: Full ASCII Code 39 0: Standard Code 39
26	1: Transmit Italian Pharmacode Check Character 0: DO NOT transmit Italian Pharmacode Check Character
27	1: Transmit CIP 39 Check Character 0: DO NOT transmit CIP 39 Check Character
28	1: Verify Interleave 25 Check Digit 0: DO NOT verify Interleave 25 Check Digit
29	1: Transmit Interleave 25 Check Digit 0: DO NOT transmit Interleave 2 Check Digit
30	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit
31	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit
32	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit
33	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit
34	Select Interleave 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern 1: Use Interleave 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern
35	Select Industrial 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern 1: Use Interleave 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern
36	Select Matrix 25 Start/Stop Pattern (to be continued)

	0: Use Industrial 25 Start/Stop Pattern 1: Use Interleave 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern
37	Codabar Start/Stop Character 0: abcd/abcd 1: abcd/tn*e 2: ABCD/ABCD 3: ABCD/TN*E
38	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character
39	MSI Check Digit Verification 0: Single Modulo 10 1: Double Modulo 10 2: Modulo 11 and Modulo 10
40	MSI Check Digit Transmission 0: the last Check Digit is not transmitted 1: both Check Digits are transmitted 2: both Check Digits are NOT transmitted
41	1: Transmit Plessey Check Characters 0: DO NOT transmit Plessey Check Characters
42	1: Convert Standard Plessey to UK Plessey 0: No conversion
43	1: Convert UPCE to UPCA 0: No conversion
44	1: Convert UPCA to EAN13 0: No conversion
45	1: Enable ISBN Conversion 0: No conversion
46	1: Enable ISSN Conversion 0: No conversion
47	1: Transmit UPCE Check Digit 0: DO NOT transmit UPCE Check Digit
48	1: Transmit UPCA Check Digit 0: DO NOT transmit UPCA Check Digit
49	1: Transmit EAN8 Check Digit (to be continued)

	0: DO NOT transmit EAN8 Check Digit
50	1: Transmit EAN13 Check Digit 0: DO NOT transmit EAN13 Check Digit
51	1: Transmit UPCE System Number 0: DO NOT transmit UPCE System Number
52	1: Transmit UPCA System Number 0: DO NOT transmit UPCA System Number
53	1: Convert EAN8 to EAN13 0: No conversion
54	1: Transmit EAN 128 Concatenation Characters 0: DO NOT transmit EAN 128 Concatenation Characters
55	1: Enable Negative Barcode 0: Disable Negative Barcode
56	0: No Read Redundancy for Scanner Port 1 1: One Time Read Redundancy for Scanner Port 1 2: Two Times Read Redundancy for Scanner Port 1 3: Three Times Read Redundancy for Scanner Port 1
57	(Not for portable terminals.)
58	1: Industrial 25 Code Length Limitation in Max/Min Length Format 0: Industrial 25 Code Length Limitation in Fixed Length Format
59	Industrial 25 Max Code Length / Fixed Length 1
60	Industrial 25 Min Code Length / Fixed Length 2
61	1: Interleave 25 Code Length Limitation in Max/Min Length Format 0: Interleave 25 Code Length Limitation in Fixed Length Format
62	Interleave 25 Max Code Length / Fixed Length 1
63	Interleave 25 Min Code Length / Fixed Length 2
64	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format
65	Matrix 25 Max Code Length / Fixed Length 1
66	Matrix 25 Min Code Length / Fixed Length 2
67	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format
68	MSI Max Code Length / Fixed Length 1
69	MSI Min Code Length / Fixed Length 2

(to be continued)

70	Scan Mode for Scanner Port 1 0: Auto Off Mode 1: Continuous Mode 2: Auto Power Off Mode 3: Alternate Mode 4: Momentary Mode 5: Repeat Mode 6: Laser Mode 7: Test Mode 8: Aiming Mode
71	(Not for portable terminals.)
72	Scanner Time-out Duration in seconds for Scanner Port 1: applicable to Aiming mode, Laser mode, Auto Off mode, and Auto Power Off mode
73	(Not for portable terminals.)
74	1: Enable RSS14 Limited 0: Disable RSS14 Limited
75	1: Enable RSS14 Expanded 0: Disable RSS14 Expanded
76	1: Enable RSS14 0: Disable RSS14
77	1: Transmit RSS14 Code ID 0: DO NOT transmit RSS14 Code ID
78	1: Transmit RSS14 Application ID 0: DO NOT transmit RSS14 Application ID
79	1: Transmit RSS14 Check Digit 0: DO NOT transmit RSS14 Check Digit
80	1: Transmit RSS14 Limited Code ID 0: DO NOT transmit RSS14 Limited Code ID
81	1: Transmit RSS14 Limited Application ID 0: DO NOT transmit RSS14 Limited Application ID
82	1: Transmit RSS14 Limited Check Digit 0: DO NOT transmit RSS14 Limited Check Digit
83	1: Transmit RSS14 Expanded Code ID 0: DO NOT transmit RSS14 Expanded Code ID
84	1: Enable AIM Telepen (Full ASCII) 0: Enable original Telepen (Numeric)

(to be continued)

85	1: Enable Telepen 0: Disable Telepen

Example READER_SETTING(1,1) ' Code 39 readability is enabled.

See Also CODE_TYPE, GET_READER_SETTING

5.9 Keyboard Wedge Commands

The portable terminals are able to send data to the host through the keyboard wedge interface by using command *SEND_WEDGE*. The *SEND_WEDGE* function is governed by a set of parameters called WedgeSetting$. Command *SET_WEDGE* is used to configure these parameters. Note only Optimus R supports wedge interface.

5.9.1 Definition of the WedgeSetting Array

WedgeSetting$ is a 3-element character array passed to SET_WEDGE to describe the characteristics of the keyboard wedge interface. In a BASIC program, WedgeSetting$ can be defined as follows.

WedgeSetting$ = Wedge_1$ + Wedge_2$ + Wedge_3$

The functions of the parameters Wedge_1$, Wedge_2$, and Wedge_3$ are described in the following subsections.

Parameter	Bit	Description
Wedge_1$	7-0	KBD / Terminal Type
Wedge_2$	7	1: Enable capital lock auto-detection 0: Disable capital lock auto-detection
Wedge_2$	6	1: Capital lock on 0: Capital lock off
Wedge_2$	5	1: Ignore alphabets' case 0: Alphabets are case sensitive
Wedge_2$	4-3	00: Normal 10: Digits are at lower position 11: Digits are at upper position

Wedge_2$	2-1	00: Normal 10: Capital lock keyboard 11: Shift lock keyboard
Wedge_2$	0	1: Use numeric keypad to transmit digits 0: Use alpha-numeric key to transmit digits
Wedge_3$	7-0	Inter-character delay

5.9.2 KBD / Terminal Type

The first element determines which type of keyboard wedge is applied. The possible value is listed as follows.

Setting Value	Terminal Type	Setting Value	Terminal Type
0	Null (Data Not Transmitted)	21	PS55 002-81, 003-81
1	PCAT (US)	22	PS55 002-2, 003-2
2	PCAT (FR)	23	PS55 002-82, 003-82
3	PCAT (GR)	24	PS55 002-3, 003-3
4	PCAT (IT)	25	PS55 002-8A, 003-8A
5	PCAT (SV)	26	IBM 3477 TYPE 4
6	PCAT (NO)	27	PS2-30
7	PCAT (UK)	28	Memorex Telex 122 Keys
8	PCAT (BE)	29	PCXT
9	PCAT (SP)	30	IBM 5550
10	PCAT (PO)	31	NEC 5200
11	PS55 A01-1	32	NEC 9800
12	PS55 A01-2	33	DEC VT220, 320, 420
13	PS55 A01-3	34	Macintosh (ADB)
14	PS55 001-1	35	Hitachi Elles
15	PS55 001-81	36	Wyse Enhance KBD (US)
16	PS55 001-2	37	NEC Astra
17	PS55 001-82	38	Unisys TO-300
18	PS55 001-3	39	Televideo 965
19	PS55 001-8A	40	ADDS 1010
20	PS55 002-1, 003-1		

For example, if the terminal type is PCAT (US), then the first element of the WedgeSetting$ can be defined as follows,

Wedge_1$ = CHR$(1)

5.9.3 Capital Lock Auto-Detection

If "*Capital Lock Auto-Detection*" is enabled, the *SEND_WEDGE* function can automatically detect the capital lock status of keyboard when the keyboard type selected is PCAT (all available languages), PS2-30, PS55, or Memorex Telex. If this is the case, the *SEND_WEDGE* function will ignore the capital lock status setting and perform auto-detection when transmitting data. If "*Capital Lock Auto-Detection*" is disabled, the *SEND_WEDGE* function will transmit alphabets according to the setting of the capital lock status.

If the keyboard type selected is not one among PCAT, PS2-30, PS55, and Memorex Telex, the *SEND_WEDGE* function will transmit the alphabets according to the setting of the capital lock status, even though the auto-detection setting is enabled.

To enable "*Capital Lock Auto-Detection*", add 128 to the value of the second element of WedgeSetting$ (Wedge_2$).

5.9.4 Capital Lock Status Setting

In order to send alphabets with correct case (upper or lower case), the *SEND_WEDGE* function must know the capital lock status of keyboard when transmitting data. Incorrect capital lock setting will result in different letter case (for example, 'A' becomes 'a', and 'a' becomes 'A').

To set "*Capital Lock ON*", add 64 to the value of the second element of WedgeSetting$ (Wedge_2$).

5.9.5 Alphabets Case

The setting of this bit affects the way the *SEND_WEDGE* function transmits alphabets. The *SEND_WEDGE* function can transmit alphabets according to their original case (case sensitive) or just ignore it. If ignoring case is selected, the *SEND_WEDGE* function will always transmit alphabets without adding shift key.

To set "*Ignore Alphabets Case*", add 32 to the value of the second element of WedgeSetting$ (Wedge_2$).

5.9.6 Digits Position

This setting can force the *SEND_WEDGE* function to treat the position of the digit keys on the keyboard differently. If this setting is set to upper, the *SEND_WEDGE* function will add shift key when transmitting digits. This setting will be effective only when the keyboard type selected is PCAT (all available language), PS2-30, PS55, or Memorex Telex. However, if the user chooses to send digits using numeric keypad, this setting is meaningless.

To set "*Lower Position*", add 16 to the value of the second element of WedgeSetting$ (Wedge_2$). To set "*Upper Position*", add 24 to the value of the second element of WedgeSetting$ (Wedge_2$).

Note: Do not set this setting unless the user is sure about the selection.

5.9.7 Shift / Capital Lock Keyboard

This setting can force the *SEND_WEDGE* function to treat the keyboard type to be a shift lock keyboard or a capital lock keyboard. This setting will be effective only when the keyboard type selected is PCAT (all available language), PS2-30, PS55, or Memorex Telex.

To set "*Capital Lock*", add 4 to the value of the second element of WedgeSetting$ (Wedge_2$). To set "*Shift Lock*", add 6 to the value of the second element of WedgeSetting$ (Wedge_2$).

Note: Do not set this setting unless the user is sure about the selection.

5.9.8 Digit Transmission

This setting instructs the *SEND_WEDGE* function which group of keys is used to transmit digits, whether to use the digit keys on top of the alphabet keys or use the digit keys on the numeric keypad.

To set "*Use Numeric Keypad to Transmit Digits*", add 2 to the value of the second element of WedgeSetting$ (Wedge_2$).

5.9.9 Inter-Character Delay

A millisecond inter-character delay, in the range of 0 to 255, can be added before transmitting each character. This is used to provide some response time for PC to process keyboard input.

For example, to set the inter-character delay to be 10 millisecond, the third element of the WedgeSetting$ can be defined as,

Wedge_3$ = CHR$(10)

5.9.10 Composition of Output String

The keyboard wedge character mapping is shown below. Each character in the output string is translated by this table when the *SEND_WEDGE* function transmits data.

	00	10	20	30	40	50	60	70	80
0		F2	SP	0	@	P	`	p	⓪
1	INS	F3	!	1	A	Q	a	q	①
2	DLT	F4	"	2	B	R	b	r	②
3	Home	F5	#	3	C	S	c	s	③
4	End	F6	$	4	D	T	d	t	④
5	Up	F7	%	5	E	U	e	u	⑤
6	Down	F8	&	6	F	V	f	v	⑥
7	Left	F9	'	7	G	W	g	w	⑦
8	BS	F10	(	8	H	X	h	x	⑧
9	HT	F11	)	9	I	Y	i	y	⑨
A	LF	F12	*	:	J	Z	j	z	
B	Right	ESC	+	;	K	[	k	{	
C	PgUp	Exec	,	<	L	\	l	|	
D	CR	CR*	-	=	M	]	m	}	
E	PgDn		.	>	N	^	n	~	
F	F1		/	?	O	_	o	Dly	

Note: (1) Dly: Delay 100 millisecond (2) ⓪~⑨: Digits of numeric keypad
(3) CR*: Enter key on the numeric keypad

The *SEND_WEDGE* function can not only transmit simple characters as above, but also provide a way to transmit combination key status, or even direct scan codes. This is done by inserting some special command codes in the output string. A command code is a character whose value is between 0xC0 and 0xFF.

0xC0 : Indicates that the next character is to be treated as scan code. Transmit it as it is, no translation required.

0xC0 | 0x01 : Send next character with Shift key.

0xC0 | 0x02 : Send next character with left Ctrl key.

0xC0 | 0x04 : Send next character with left Alt key.

0xC0 | 0x08 : Send next character with right Ctrl key.

0xC0 | 0x10 : Send next character with right Alt key.

0xC0 | 0x20 : Clear all combination status key after sending the next character.

For example, to send [A] [Ctrl-Insert] [5] [scan code 0x29] [Tab] [2] [Shift-Ctrl-A] [B] [Alt-1] [Alt-2-Break] [Alt-1] [Alt-3], the following characters are inserted into the string supplied to the *SEND_WEDGE* function.

0x41, 0xC2, 0x01, 0x35, 0xC0, 0x29, 0x09, 0x32, 0xC3, 0x41, 0x42, 0xC4, 0x31 0xE4, 0x32, 0xC4, 0x31, 0xC4, 0x33

Please note that, the scan code 0x29 is actually a space for PCAT, Alt-12 is a form feed character, and Alt-13 is an Enter. The break after Alt-12 is necessary, if omitted the characters will be treated as Alt-1213 instead of Alt-12 and Alt-13.

The following instructions can be called in the BASIC program to send the above string through the keyboard wedge interface.

```
...

Data_1$ = CHR$(65) + CHR$(194) + CHR$(1) + CHR$(53) + CHR$(192) + CHR$(41)

Data_2$ = CHR$(9) + CHR$(50) + CHR$(195) + CHR$(65) + CHR$(66)

Data_3$ = CHR$(196) + CHR$(49) + CHR$(228) + CHR$(50) + CHR$(196) + CHR$(49)

Data_4$ = CHR$(196) + CHR$(51)

DataString$ = Data_1$ + Data_2$ + Data_3$ + Data_4$

SEND_WEDGE (DataString$)
```

SEND_WEDGE

Purpose To send data to the host via keyboard wedge interface.

Syntax SEND_WEDGE(*DataString$*)

Remarks "*DataString$*" is the data string to be sent via the keyboard wedge interface.

Example ...

```
DataString$ = CHR$(9) + "TESTING" +          ' [Tab] + "TESTING" + [Tab]
CHR(9)
SEND_WEDGE (DataString$)
...
```

See Also SET_WEDGE, WEDGE_READY

SET_WEDGE

Purpose To configure the keyboard wedge interface.

Syntax SET_WEDGE(*WedgeSetting$*)

Remarks "*WedgeSetting$*" is a 3-element character array describing the characteristics of the keyboard wedge interface.

Example ...

```
Wedge_1$ = CHR$(1)                    ' terminal type: PCAT(US)
Wedge_2$ = CHR$(1)
' auto-detection disabled, capital lock off, case sensitive
' use numeric keypad to transmit digits
Wedge_3$ = CHR$(5)                    ' inter-char-delay: 5 ms
WedgeSetting$ = Wedge_1$ + Wedge_2$ + Wedge_3$
SET_WEDGE(WedgeSetting$)
SEND_WEDGE (DataString$)
...
```

See Also SEND_WEDGE, WEDGE_READY

WEDGE_READY

Purpose To check if the keyboard wedge cable is well connected and ready to send data.

Syntax *state%* = WEDGE_READY

Remarks The return value is 1 if it is ready for sending data; otherwise, it returns 0.

Note that it takes approximately 110 ms to detect the status of the keyboard wedge connection. Therefore, for continuous and fast data transmission, simply call this function once. DO NOT call this function repeatedly.

Example

```
IF (WEDGE_READY = 1) THEN
   ...
   SEND_WEDGE (DATA$)
   ...
END IF
   ...
```

See Also SEND_WEDGE, SET_WEDGE

5.10 Buzzer Commands

This section describes the commands relating to the beeper.

BEEP

Purpose To produce audible signals according to a specified beeping sequence.

Syntax BEEP(*freq%, duration% {, freq%, duration%}*)

Remarks "*freq%*" is an integer variable, indicating the value of beeping frequency (Hz). Suggested frequency for the beeper ranges from 1 kHz to 6 kHz.

"*duration%*" is an integer variable, indicating the value of beeping duration, which is specified in units of 10 milliseconds.

Up to eight frequency-duration pairs can be assigned in a BEEP command. If the value of the frequency is 0, the beeper will not sound during the time duration.

Example

```
ON READER(1) GOSUB BcrDATA_1
   ...
BcrData_1:
   BEEP(2000, 10, 0, 10, 2000, 10)
   ...
   RETURN
```

See Also STOP BEEP

KEY_CLICK

Purpose To enable/disable the key click sound.

Syntax KEY_CLICK(*status%*)

Remarks "*status%*" is an integer variable, indicating the key click status.

Value	Meaning
0	Mute
1 ~ 5	Each represents a different tone

The key click is enabled by default.

Example
```
KEY_CLICK(0)                    ' disable the key click
```

STOP BEEP

Purpose To terminate the beeping.

Syntax STOP BEEP

Remarks The STOP BEEP statement terminates the beeping immediately if there is a beeping sequence in progress.

Example
```
BEEP(2000,0)
ON KEY(1) GOSUB StopBeep
PRINT "Press F1 to stop the beeper."
…
StopBeep:
STOP BEEP
RETURN
```

See Also BEEP

5.12 Calendar and Timer Commands

This section describes the commands relating to the calendar and timer. The system date and time are maintained by the calendar chip, and they can be retrieved from or set to the calendar chip by the *DATE$* and *TIME$* functions. A backup rechargeable battery keeps the calendar chip running even when the power is turned off.

Commands for triggering the HOUR_SHARP event, the MINUTE_SHARP event, and the TIMER event: *OFF HOUR_SHARP, OFF MINUTE_SHARP, OFF TIMER, ON HOUR_SHARP GOSUB..., ON MINUTE_SHARP GOSUB..., ON TIMER GOSUB...*

Up to five timers can be set by the *ON TIMER... GOSUB...* command for the "TIMER Event Trigger".

> Note: The system time variable TIMER is maintained by CPU timers and has nothing to do with this calendar chip. Accuracy of this time variable depends on the CPU clock and is not suitable for precise time manipulation. Besides, it is reset to 0 upon powering up (as a cold start).

DATE$

Purpose To set or to get the current date.

Syntax DATE$ = *X$*

*Y$* = DATE$

Remarks DATE$ = *X$*, to set the current date.

"*X$*" is a string variable in the form of "yyyymmdd".

*Y$* = DATE$, to get the current date, in the form of "yyyymmdd".

"*Y$*" is a string variable to be assigned to the result.

The BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to DATE$. The user is obliged to check the format and contents.

Example

```
DATE$ = "20000103"          ' set the system date to 2000/01/03
Today$ = DATE$              ' assign the current date to Today$
PRINT Today$                ' Today$ = "20000103"
```

See Also DAY_OF_WEEK, TIME$

DAY_OF_WEEK

Purpose To get the day of the week.

Syntax *A%* = DAY_OF_WEEK

Remarks "*A%*" is an integer variable to be assigned to the result.

A value of 1 to 7 represents Monday to Sunday respectively.

Example

```
ON DAY_OF_WEEK GOSUB 100, 200, 300, 400, 500, 600, 700
...
100
  PRINT "Today is Monday."
  RETURN
200
  PRINT "Today is Tuesday."
  RETURN
300
  PRINT "Today is Tuesday."
RETURN
```

See Also DATE$, TIME$

TIME$

Purpose To set or to get the current time.

Syntax TIME$ = *X$*

*Y$* = TIME$

Remarks TIME$ = *X$*, to set the current time.

"*X$*" is a string variable in the form of "hhmmss".

*Y$* = TIME$, to get the current time, in the form of "hhmmss".

"*Y$*" is a string variable to be assigned to the result.

The BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to TIME$. The user is obliged to check the format and contents.

Example

```
TIME$ = "112500"                ' set the system time to 11:25:00
CurrentTime$ = TIME$            ' assign the current to CurrentTime$
PRINT CurrentTime$              ' CurrentTime$ = "112500"
```

See Also DATE$, DAY_OF_WEEK

TIMER

Purpose To return the number of seconds since the terminal is powered on.

Syntax *A&* = TIMER

Remarks "*A&*" is a long integer variable to be assigned to the result.

Note that the TIMER is a read-only function. The system timer cannot be set by using this function.

Example

```
StartTime& = TIMER
...
Loop:
  IF EndTime& <> TIMER THEN
    EndTime& = TIMER
    TimerElapsed& = EndTime& - StartTime&
    CLS
    PRINT TimerElapsed&
    IF TimerElapsed& > 100 THEN GOTO NextStep
  END IF
  GOTO Loop
NextStep:
…
```

See Also OFF TIMER, ON TIMER GOSUB…

WAIT

Purpose To put the system on hold for a specified duration. In the interval, the system will be running in a rather low power consumption mode.

Syntax WAIT(*duration%*)

Remarks "*duration%*" is a positive integer variable, indicating the time duration for a hold. This argument is specified in units of 5 milliseconds.

When the application is waiting for events in a loop, the power consumption will be dramatically reduced by calling this function.

Example

```
PRINT " BASIC"                    ' the system is on hold for 1 second
WAIT (200)
```

5.13 LED Command

The LED can be used to indicate terminal status, for example, good read or no good when scanning a barcode.

LED

Purpose To specify the LED lighting behavior.

Syntax LED(*number%, mode%, duration%*)

Remarks "*number%*" is a positive integer variable, indicating the LED number.

Value	Meaning
1	Red LED
2	Green LED

"*mode%*" is an integer variable, indicating the digital output mode. The values of the mode and their interpretation are listed below.

Value	Meaning
1	Turn on the LED for the specific duration and then go back off.
2	Turn off the LED for the specific duration and then go back on.
3	Flash the LED for a specific duration repeatedly. The flashing period equals 2X*duration*.

"*duration%*" is an integer variable, specifying a period of time in units of 10 milliseconds. A value of 0 in this argument will keep the LED in the specific state indefinitely.

Example

```
ON READER(1) GOSUB BcrData_1
  ...
BcrData_1:
  BEEP(2000,5)
  LED(2, 1, 5)                          ' GOOD READ LED for 520
  Data$ = GET_READER_DATA$(1)
  ...
```

5.14 Keypad Commands

All the portable terminals provide a built-in keypad for data input. This section describes the commands relating to the keypad operation.

Commands for triggering the ESC event and the KEY event: *OFF ESC, OFF KEY, ON ESC GOSUB..., ON KEY GOSUB...*

ALPHA_LOCK

Purpose To set the input mode and key lock status.

Syntax ALPHA_LOCK(*status%*)

Remarks "*status%*" is an integer variable, indicating the alpha-input status.

Possible settings for the 7xx series:

Value	Meaning
0	Unlocked, numeric mode by default.
1	Locked in alpha mode.
2	Locked in numeric mode.

Possible settings for the 8xxx series:

Value	Meaning
0	Unlocked, numeric mode by default.
1	Unlocked, upper case alpha-input by default.
2	Locked in numeric mode.
3	Unlocked, lower case alpha-input by default.
4	Unlocked, function mode by default. (8000 only)
5	Locked in upper case alpha mode.
6	Locked in lower case alpha mode.
7	Locked in function mode. (8000 only)

Example
```
ALPHA_LOCK(1)
...
```

See Also GET_ALPHA_LOCK

CLR_KBD

Purpose To clear the keyboard buffer.

Syntax CLR_KBD

Remarks By calling this function, the data queuing in the keyboard buffer will be cleared.

Example
```
CLR_KBD
ON KEY(1) GOSUB KeyData_1
...
```

See Also INKEY$, PUTKEY

FUNCTION_TOGGLE — Optimus

Purpose To set the state of function toggle.

Syntax FUNCTION_TOGGLE(*status%*)

Remarks "*status%*" is an integer variable, indicating the state of the function toggle.

Value	Meaning
0	Off
1	On

Example
```
FUNCTION_TOGGLE(1)          ' turn on the function toggle
```

GET_ALPHA_LOCK

Purpose To get alpha-lock status.

Syntax *A%* = GET_ALPHA_LOCK

Remarks "*A%*" is an integer variable to be assigned to the result.

Example
```
Alpha_lock% = GET_ALPHA_LOCK
```

See Also ALPHA_LOCK

INKEY$

Purpose To read one character from the keyboard buffer and then remove it.

Syntax *X$* = INKEY$

Remarks "*X$*" is a string variable to be assigned to the character read.

It can be used with menu operation to detect a shortcut key being pressed, or with touch screen operation to detect a touched item.

Example

```
...
PRINT "Initialize System (Y/N)?"
Loop:
  KeyData$ = INKEY$
  IF KeyData$ = "" THEN
    GOTO Loop
  ELSE IF KeyData$ = "Y" THEN
    GOTO Initialize
  …
```

See Also CLR_KBD , PUTKEY

INPUT

Purpose To read user input from the keypad, store the input data in a variable, and print it to the display.

Syntax INPUT *variable*

Remarks "*variable*" is a numeric or string variable that will receive the input data. The data entered must match the data type of the variable.

Note that the use of ENTER key is necessary to end with an input session.

Example

```
INPUT String$                    ' input a string variable
PRINT String$
INPUT Number%                    ' input a numeric variable
PRINT Number%
```

INPUT_MODE

Purpose To set the display mode of the input data.

Syntax INPUT_MODE(*mode%*)

Remarks "*mode%*" is an integer variable, indicating the input mode.

Value	Meaning
0	Nothing will be displayed on the LCD.
1	The input characters will be displayed on the LCD (default).
2	"*" will be displayed instead of the input characters.

Example
```
LOCATE 1,1
INPUT_MODE(1)
INPUT Login$
LOCATE 2,1
INPUT_MODE(2)
INPUT Password$
```

5.15 LCD Commands

This section describes the commands relating to the liquid crystal display (LCD).

5.15.1 Graphic Display

Almost all the portable terminals are equipped with a graphic display allowing the programmer to show an image on it. A coordinated system is also used for governing the cursor movement. The coordinates of the top left position are assigned with (1,1) while those of the bottom right position depends on the varying size of both screen and font.

BACK_LIGHT_DURATION

Purpose To determine how long the backlight will last once being turned on.

Syntax BACK_LIGHT_DURATION(*N%*)

Remarks "*N%*" is an integer variable, indicating a period of time in units of 1 second.

Example
```
BACK_LIGHT_DURATION(20)          ' backlight lasts for 20 seconds
```

See Also LCD_CONTRAST, SET_VIDEO_MODE

CLR_RECT

Purpose To clear a rectangular area.

Syntax CLR_RECT(*x%, y%, size_x%, size_y%*)

Remarks "*x%*" is the x coordinate of the upper left point of the rectangular area.

"*y%*" is the y coordinate of the upper left point of the rectangular area.

"*size_x%*" is the width of the rectangular area in pixels.

"*size_y%*" is the height of the rectangular area in pixels.

Example CLR_RECT(1,1,20,20)

See Also CLS, FILL_RECT

CLS

Purpose To clear the contents on the LCD.

Syntax CLS

Remarks After executing this command, whatever is being shown on the LCD will be erased and the cursor will be move to (1,1).

Example

```
ON TIMER(1,200) GOSUB ClearScreen
...
ClearScreen:
  OFF TIMER(1)                    ‘ TIMER(1) = 2 second
  CLS
  RETURN
```

See Also CLR_RECT, PRINT

CURSOR

Purpose To turn on/off the cursor indication on the LCD.

Syntax CURSOR(*status%*)

Remarks "*status%*" is an integer variable, indicating the cursor status.

Value	Meaning
0	The cursor indication is off.
1	The cursor indication is on.

Example CURSOR(0)

See Also CURSOR_X, CURSOR_Y, LOCATE

CURSOR_X

Purpose To get the current x-coordinate of the cursor.

Syntax *X%* = CURSOR_X

Remarks "*X%*" is an integer variable to be assigned to the column position of the cursor.

Example

```
ON READER(1) GOSUB BcrData_1
...
BcrData_1:
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  Pre_X% = CURSOR_X
  Pre_Y% = CURSOR_Y
  Locate 8, 1
  PRINT Data$
  Locate Pre_Y%, Pre_X%
RETURN
```

See Also CURSOR, CURSOR_Y, LOCATE

CURSOR_Y

Purpose To get the current y-coordinate of the cursor.

Syntax "*Y%*" = CURSOR_Y

Remarks "*Y%*" is an integer variable to be assigned to the column position of the cursor.

Example

```
ON READER(1) GOSUB BcrData_1
...
BcrData_1:
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  Pre_X% = CURSOR_X
  Pre_Y% = CURSOR_Y
  Locate 8, 1
  PRINT Data$
  Locate Pre_Y%, Pre_X%
  RETURN
```

See Also CURSOR, CURSOR_X, LOCATE

FILL_RECT

Purpose To fill a rectangular area.

Syntax FILL_RECT(*x%, y%, size_x%, size_y%*)

Remarks "*x%*" is the x coordinate of the upper left point of the rectangular area.
"*y%*" is the y coordinate of the upper left point of the rectangular area.
"*size_x%*" is the width of the rectangular area in pixels.
"*size_y%*" is the height of the rectangular area in pixels.

Example FILL_RECT(1,1,20,20)

See Also CLR_RECT

GET_IMAGE

Purpose To read a bitmap image or capture signature of a rectangular area on the LCD.

Syntax *DataCount%* = GET_IMAGE(*file_index%, x%, y%, size_x%, size_y%*)

Remarks "*DataCount%*" is an integer variable to be assigned to the result; it is the total data count stored in the specified transaction file.

"*file_index%*" is an integer variable in the range of 1 to 6, indicating which transaction file is to store the bitmap data.

"*x%*" is an integer variable, indicating the x coordinate of the upper left point of the rectangular area.

"*y%*" is an integer variable, indicating the y coordinate of the upper left point of the rectangular area.

"*size_x%*" is an integer variable, indicating the width of the rectangular area in pixels.

"*size_y%*" is an integer variable, indicating the height of the rectangular area in pixels.

Example GET_IMAGE(3,12,32,60,16)

See Also GET_TRANSACTION_DATA$, GET_TRANSACTION_DATA_EX$, SET_SIGNAREA, SHOW_IMAGE

ICON_ZONE_PRINT

Purpose To enable or disable the printing of characters in the icon area.

Syntax ICON_ZONE_PRINT(*status%*)

Remarks "*status%*" an integer variable, indicating the printing status of the icon area.

Value	Meaning
0	The printing in the icon area is not allowed (default).
1	The printing in the icon area is allowed.

Take CPT-711 for example.

Its LCD has 128x64 pixels, but the default printable area is 120x64 pixels. That is, it can have

- 20x8 characters for small font(6x8 dots per char), or

- 15x4 characters for large font (8x16 dots per char).

The icon area is the right-most 8x64 pixels that can only be accessed by graphic commands by default. If the icon area is set to be printable, then CPT-711 can have

- 21 characters per line for small font, or

- 16 characters per line for large font.

Note that the system may still show the battery and alpha icons of this icon area, even if is set to be printable for the user program.

Example ICON_ZONE_PRINT(1) ' allow the printing of the icon area

See Also PRINT, SHOW_IMAGE

LCD_CONTRAST

Purpose To set the contrast level for the LCD.

Syntax LCD_CONTRAST(*N%*)

Remarks "*N%*" is an integer variable, indicating the contrast level in the range of 1 to 8. The higher value, the stronger contrast.

Example
```
LCD_CONTRAST(4)          ' set the LCD contrast to level 4 (medium contrast)
```

See Also BACK_LIGHT_DURATION, SET_VIDEO_MODE

LOCATE

Purpose To move the cursor to a specified location on the LCD.

Syntax LOCATE *row%, col%*

Remarks "*row%*" is an integer variable, indicating the new row position of the cursor.

"*col%*" is an integer variable, indicating the new column position of the cursor.

Example
```
LOCATE 1,1               ' move the cursor to the top left of the LCD
```

See Also CURSOR, CURSOR_X, CURSOR_Y

PRINT

Purpose To display data on the LCD.

Syntax PRINT *expression*[{,|;[*expression*]}]

Remarks "*expression*" may be numeric or string expression.

The position of each printed item is determined by the punctuation used to separate items in the list. In the list of expression, a comma causes the next character to be printed after the last character with a blank space. A semicolon causes the next character to be printed immediately after the last character. If the list of expressions terminates without a comma or semicolon, a carriage return is printed at the end of the line.

Example
```
LOCATE 1,1
PRINT String$(20,"")                    ' clear the whole line
LOCATE 1,1
A = 5
PRINT A, "square is "; A*A
```

See Also CLS, ICON_ZONE_PRINT, SHOW_IMAGE

SHOW_IMAGE

Purpose To show a bitmap image on the LCD.

Syntax SHOW_IMAGE(*x%, y%, size_x%, size_y%, image$*)

Remarks "*x%*" is an integer variable, indicating the x coordinate of the upper left point of the rectangular area.

"*y%*" is an integer variable, indicating the y coordinate of the upper left point of the rectangular area.

"*size_x%*" is an integer variable, indicating the width of the rectangular area in pixels.

"*size_y%*" is an integer variable, indicating the height of the rectangular area in pixels.

"*image$*" is a string variable, containing the bitmap data of the image.

The user needs to allocate a string variable to store the bitmap data of the image. The string begins with the top row of pixels. Each row begins with the left-most pixels. Each bit of the bitmap represents a single pixel of the image. If the bit is set to 1, the pixel is marked; if it is 0, the pixel is unmarked. The first pixel in each row is represented by the least significant bit of the first byte in each row. If the image is wider than 8 pixels, the ninth pixel in each row is represented by the least significant bit of the second byte in each row.

Below is an example of the company logo and the string variable - icon$, which is used to store the bitmap data.

Example

```
icon_1$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(248)+chr$(255)+chr$(7)
icon_2$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(8)+chr$(0)+chr$(4)
icon_3$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_4$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_5$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_6$ = chr$(192)+chr$(3)+chr$(0)+chr$(0)+chr$(250)+chr$(255)+chr$(5)
icon_7$ =
chr$(96)+chr$(214)+chr$(201)+chr$(59)+chr$(250)+chr$(142)+chr$(5)
icon_8$ = chr$(48)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)
icon_9$ = chr$(16)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)
icon_10$ =
chr$(16)+chr$(208)+chr$(249)+chr$(59)+chr$(186)+chr$(139)+chr$(5)
icon_11$ = chr$(48)+chr$(84)+chr$(72)+chr$(24)+chr$(58)+chr$(104)+chr$(5)
icon_12$ = chr$(96)+chr$(86)+chr$(72)+chr$(40)+chr$(186)+chr$(107)+chr$(5)
icon_13$ =
chr$(192)+chr$(83)+chr$(200)+chr$(75)+chr$(130)+chr$(139)+chr$(5)
icon_14$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(7)
icon_15$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)
icon_16$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)
show_image(2, 0, 56, 1, icon_1$)
show_image(2, 1, 56, 1, icon_2$)
show_image(2, 2, 56, 1, icon_3$)
show_image(2, 3, 56, 1, icon_4$)
show_image(2, 4, 56, 1, icon_5$)
show_image(2, 5, 56, 1, icon_6$)
show_image(2, 6, 56, 1, icon_7$)
show_image(2, 7, 56, 1, icon_8$)
show_image(2, 8, 56, 1, icon_9$)
show_image(2, 9, 56, 1, icon_10$)
show_image(2, 10, 56, 1, icon_11$)
show_image(2, 11, 56, 1, icon_12$)
show_image(2, 12, 56, 1, icon_13$)
show_image(2, 13, 56, 1, icon_14$)
show_image(2, 14, 56, 1, icon_15$)
show_image(2, 15, 56, 1, icon_16$)
```

See Also GET_IMAGE, ICON_ZONE_PRINT, PRINT

WAIT_HOURGLASS — Optimus

Purpose To show a moving hourglass on the LCD.

Syntax WAIT_HOURGLASS(*x%, y%, type%*)

Remarks Call this function constantly to maintain its functionality. There are five different patterns of an hourglass that take turn to show on the LCD within a certain period to symbolize the passage of time. The time factor is decided by programming but never less than two seconds.

"*x%*" is an integer variable, indicating the x coordinate of the upper left point of a hourglass.

"*y%*" is an integer variable, indicating the y coordinate of the upper left point of a hourglass.

"*type%*" is an integer variable, indicating the size of a hourglass.

Value (type%)	Meaning
1	24 x 23 pixels
2	8 x 8 pixels

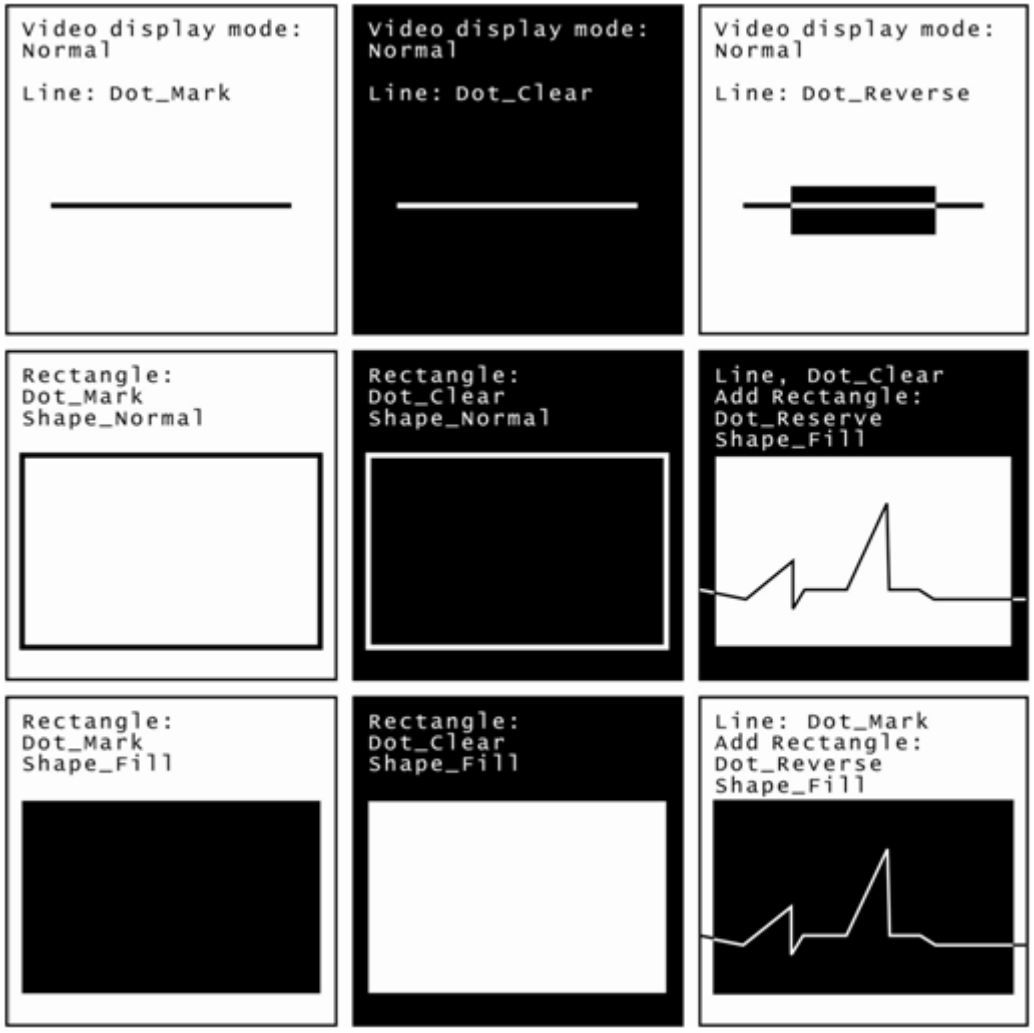
Example

```
WAIT_HOURGLASS(68,68,1)        ' show a 24 x 23 pixels hourglass at (68,68)
```

Factors to Monochrome Graphic

A monochrome graphic has three factors:

Key Factor	Parameter	Value	Meaning
Video Display mode:	mode%	0	Normal mode
		1	Reverse mode
Pixel state:	mode%	-1	Dot_Reverse
		0	Dot_Clear
		1	Dot_Mark
Shape state:	type%	0	Shape_Normal
		1	Shape_Fill



SET_VIDEO_MODE

Purpose To set the display mode for the LCD.

Syntax SET_VIDEO_MODE(*mode%*)

Remarks "*mode%*" is an integer variable, indicating the display mode. (0 for Normal, 1 for Reverse)

Example

```
SET_VIDEO_MODE(1)          ' this string will be printed in reverse mode
PRINT " terminals"
```

See Also BACK_LIGHT_DURATION, LCD_CONTRAST

5.15.2 Special Font Files

There are two standard fonts, namely the *6x8* and *8x16* fonts. However, the portable terminals are capable of displaying special characters, such as the Chinese and Japanese characters, providing that the specific font file has been downloaded to the terminals. Refer to section 2.1 for details.

SELECT_FONT

Purpose To select a font size for the LCD to display.

Syntax SELECT_FONT(*font%*)

Remarks "*font%*" is an integer variable, indicating the font size.

If you select 4, the font size for single byte characters will be 6x12, while it will still take 12x12 for double byte characters. It thus provides flexibility for full/half width Katakana (Japanese), Chinese, and alphanumeric, etc.

Value	Meaning
1	Font size 6x8
2	Font size 8x16
3	Font size 16x24 (for Thai)
4	Font size 6x12 (for TC, SC, JP)
5	Font size 12x12 (for TC, SC, JP)

Example SELECT_FONT(2)

5.16 Battery Commands

This section describes the commands relating to the battery power.

BACKUP_BATTERY

Purpose To get the voltage level of the backup battery.

Syntax A% = BACKUP_BATTERY

Remarks "A%" is an integer variable to be assigned to the result.

The voltage level of the backup battery is returned in units of milli-volt (mV). The backup battery is used to backup the SRAM and keep the calendar chip running, even when the power is off. The backup battery would be considered as "Battery Low" when the BACK_BATTERY is lower than 2900 mV. That means the SRAM and the calendar chip may lose their data at any time thereafter, if the battery is not recharged or replaced.

Example

```
CheckBackupBattery:
  IF BACKUP_BATTERY < BATTERY_LOW% THEN
  BEEP(2000,30)
  CLS
  PRINT "Backup Battery needs to be replaced!"
  Loop:
    GOTO Loop
  END IF
```

See Also MAIN_BATTERY

MAIN_BATTERY

Purpose To get the voltage level of the main battery.

Syntax *A%* = MAIN_BATTERY

Remarks "*A%*" is an integer variable to be assigned to the result.

The voltage level of the main battery is returned in units of milli-volt (mV). The main battery is the power source for the system operation. The main battery would be considered as "Battery Low" when the MAIN_BATTERY is lower than 2200 mV. That means the basic operations may still be running, but some functions that consume high power may be disabled.

Example

```
  BATTERY_LOW% = 2200
CheckMainBattery:
  IF MAIN_BATTERY < BATTERY_LOW% THEN
  BEEP(2000,30)
  CLS
  PRINT "Main Battery needs to be recharged!"
  Loop:
    GOTO Loop
  END IF
```

See Also BACKUP_BATTERY

5.17 Communication Ports

There are at least two communication ports on each terminal, namely *COM1* and *COM2*. The user has to call the *SET_COM_TYPE* function to set up the communication type for the COM ports before using them.

5.17.1 Port Mapping

The following table shows the mapping of the communication ports.

Model#	COM1	COM2	COM3	COM4
Optimus S	Serial IR, IrDA	RF	N/A	N/A
Optimus R	RS-232, Serial IR, IrDA	RF	N/A	N/A

Specifying which type of interface is to be used, the user can use the same commands to open, close, read, and write the data via different interfaces: RS-232, Serial IR, IrDA, Bluetooth (SPP/DUN), or WLAN.

Commands for triggering the COM event: *OFF COM, ON COM GOSUB...*

CLOSE_COM

Purpose To disable a specified communication port.

Syntax CLOSE_COM(*N%*)

Remarks "*N%*" is an integer variable, indicating which communication port is to be disabled.

Example CLOSE_COM(2)

See Also OPEN_COM, READ_COM$, SET_COM_TYPE, WRITE_COM

OPEN_COM

Purpose To enable a specified communication port.

Syntax OPEN_COM(*N%*)

Remarks "*N%*" is an integer variable, indicating which communication port is to be enabled.

Example OPEN_COM(1)

See Also CLOSE_COM, OFF COM, ON COM GOSUB…, READ_COM$, SET_COM_TYPE, WRITE_COM

READ_COM$

Purpose To read data from a specified communication port.

Syntax *A$* = READ_COM$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the data.

"*N%*" is an integer variable, indicating from which communication port the data is to be read. If the buffer of the communication port is empty, an empty string will be returned.

Example

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$,1)
  DBFNum% = VAL(MID$(Cmd$,2,1))
  IDFNum% = VAL(MID$(Cmd$,3,1))
  CardID$ = RIGHT$(Cmd$,LEN(Cmd$)-3)
  IF CmdIdentifier$ = "-" THEN
    DEL_RECORD(DBFNum%,IDFNum%)
  ELSE
    ...
```

See Also CLOSE_COM, OFF COM, ON COM GOSUB…, OPEN_COM, SET_COM_TYPE, WRITE_COM

SET_COM_TYPE

Purpose To assign the communication type to a specified COM port.

Syntax SET_COM_TYPE(N%, type%)

Remarks "*N%*" is an integer variable, indicating which communication port is to be set.

"*type%*" is an integer variable, indicating the type of interface.

Value (type%)	Meaning
1	Direct RS-232
2	N/A
3	Serial IR (via IR transceiver)

4	Standard IrDA
5	RF, Bluetooth (SPP/DUN)
6	GSM_SMS
7	GSM_Modem

This function needs to be called BEFORE opening a COM port. Note that the COM port mapping is different for each model of terminal, and, a COM port may not support all the communication types.

Example SET_COM_TYPE(1,3) ' set COM1 of 8300 to serial IR communication

See Also CLOSE_COM, OPEN_COM, READ_COM$, SET_COM, WRITE_COM

WRITE_COM

Purpose To send data string to the host through a specified communication port.

Syntax WRITE_COM(*N%, A$*)

Remarks "*N%*" is an integer variable, indicating which communication port the data is to be sent to.

"*A$*" is a string variable, representing the character string to be sent.

Example

```
ON READER(1) GOSUB BcrData_1
...
BcrData_1:
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  WRITE_COM(1, Data$)
  ...
```

See Also CLOSE_COM, OPEN_COM, READ_COM$, SET_COM_TYPE

The rest communication-related commands are divided into different types and are described separately as follows.

5.17.2 RS-232, Serial IR and IrDA Communications

Baud rate:	Eight options - 115200, 76800, 57600, 38400, 19200, 9600, 4800, 2400
Data bits:	7 or 8
Parity:	Even, Odd, or None
Stop bit:	1
Flow Control (RS-232 only):	RTS/CTS, XON/XOFF, or None

COM_DELIMITER

Purpose To change the delimiter of COM port sending and receiving string.

Syntax COM_DELIMITER(*N%, C%*)

Remarks "*N%*" is an integer variable, indicating which communication port is to be set.

"*C%*" is an integer variable, standing for the ASCII code of the delimiter character, in the range of 0 to 255. If it is negative, no delimiter will be applied.

Example
```
COM_DELIMITER(1,10)                ' use the Line-feed character as delimiter
```

GET_CTS

Purpose To get the CTS level from the direct RS-232 port.

Syntax *A%* = GET_CTS(*N%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
0	CTS OFF
1	CTS ON

"*N%*" is an integer variable, indicating which communication port the user intends to get the CTS level for.

Example
```
A% = GET_CTS(1)
```

See Also SET_RTS

IRDA_STATUS

Purpose To check the IrDA connection status or transmission status.

Syntax *A%* = IRDA_STATUS(*N%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
0	NG
1	OK

"*N%*" is an integer variable, indicating the action to take.

Value	Meaning
0	To check IrDA connection status.
1	To check whether data being transmitted successfully or not.

Example

```
CLS
SET_COM_TYPE(2,4)                ' set COM2 of 711 as IrDA port
OPEN_COM(2)

Loop1:
IF (IRDA_STATUS(0) = 1) THEN     ' check connection
   BEEP(4400,5)
ELSE
   GOTO Loop1
END IF

WRITE_COM(2, "My Data")
IF (IRDA_STATUS(1) = 1) THEN     ' check transmission
   PRINT "Write OK"
ELSE
   PRINT "Write NG"
END IF

CLOSE_COM(2)
```

See Also IRDA_TIMEOUT

IRDA_TIMEOUT

Purpose To set the timeout for IrDA connection.

Syntax IRDA_TIMEOUT(*N%*)

Remarks "*N%*" is an integer variable in the range of 1 to 8, indicating a specified period of time.

Value	Meaning
1	3 sec
2	8 sec
3	12 sec
4	16 sec
5	20 sec
6	25 sec
7	30 sec
8	40 sec

Example
```
IRDA_TIMEOUT(7)                    ' set timeout to 30 seconds
```

See Also IRDA_STATUS

SET_COM

Purpose To set parameters for a specified communication port.

Syntax SET_COM(*N%, Baudrate%, Parity%, Data%, Handshake%*)

Remarks

Parameters	Values	Remarks
N%	1 or 2	Indicates which COM port is to be set.
Baudrate%	1: 115200 bps 2: 76800 bps* 3: 57600 bps 4: 38400 bps 5: 19200 bps 6: 9600 bps 7: 4800 bps* 8: 2400 bps*	Specifies the baud rate of the COM port. * (asterisk): Not applicable to Serial IR.

Parity%	1: None 2: Odd 3: Even	Specifies the parity of the COM port.
Data%	1: 7 data bits 2: 8 data bits	Specifies the data bits of the COM port.
Handshake%	1: None 2: CTS/RTS 3: XON/XOFF	Specifies the method of flow control for the COM port.

Example SET_COM(1,1,1,2,1) ' COM1, 115200, None, 8, No handshake

See Also CLOSE_COM, OPEN_COM, READ_COM$, SET_COM_TYPE, WRITE_COM

SET_RTS

Purpose To set the RTS level for the direct RS-232 port.

Syntax SET_RTS(*N1%, N2%*)

Remarks "*N1%*" is an integer variable, indicating for which communication port to set the RTS level.

"*N2%*" is an integer variable, indicating the RTS state.

Value	Meaning
0	OFF or in the "mark" state.
1	ON or in the "space" state.

Example SET_RTS(1,1) ' set COM1 RTS to the "mark" state

See Also GET_CTS

5.18 RF Communications

This section describes the BASIC functions and statements relating to Radio Frequency communications. These command sets are only applicable to the specific models as described below.

5.18.1 RF Models & Specifications

Below are the two types of RF communications and their specifications. A base refers to a base station, either for Narrow Band (433 MHz) or Spread Spectrum (2.4 GHz).

Product Family	433 MHz	2.4 GHz
RF, embedded	Narrow Band	Spread Spectrum

- **Narrow Band (also referred as 433 MHz RF)**

Frequency Range:	433.12 ~ 434.62 MHz
Data Rate:	9600 bps
Programmable Channels:	4
Coverage:	200 meters line-of-sight
Maximum Output Power:	10 mW (10 dbm)
Spread Spectrum:	None
Modulation:	FSK (Frequency Shift Keying)

- **Spread Spectrum (also referred as 2.4 GHz RF)**

Frequency Range:	2.4000 ~ 2.4835 GHz, unlicensed ISM Band
Data Rate:	19200 bps
Programmable Channels:	6
Coverage:	1000 meters line-of-sight
Maximum Output Power:	100 mW
Spread Spectrum:	Frequency Hopping Spread Spectrum (FHSS)
Modulation:	GFSK
Frequency Control:	Direct FM

Note: All specifications are subject to change without prior notice.

5.18.2 RF System Requirements

Base to Host:	RS-232 connection
Base Baud Rate:	Up to 115,200 bps
Base to Base:	RS-485 connection
Number of Bases per System:	Maximum 16

➢ 433 MHz RF System

Number of Terminals per Base:	Maximum 15
Number of Terminals per System:	Maximum 45

➢ 2.4 GHz RF System

Number of Terminals per Base:	Maximum 99
Number of Terminals per System:	Maximum 99

➢ IDs and Groups

An ID to a terminal or a base is like a name to a person. Each terminal/base in the same RF system must have a unique ID so that the system will work properly.

For the 433 MHz RF system, up to 45 terminals and 16 bases can be supported by one system. The valid ID ranges from 1 to 45 for terminals, and 1 to 16 for bases. To support all 45 terminals, the 433 MHz RF bases need to be configured to 3 groups. Each group, as well as each base, can support up to 15 terminals.

- Base IDs (433 MHz): 01 ~ 16
- Terminal IDs (433 MHz): 01 ~ 45 (3 groups)
  01 ~ 15 supported by Group 1 bases
  16 ~ 30 supported by Group 2 bases
  31 ~ 45 supported by Group 3 bases

For the 2.4 GHz RF system, up to 99 terminals and 16 bases can be supported by one system, and they all belong to one group. The valid ID ranges from 1 to 99 for terminals, and 1 to 16 for bases.

- Base IDs (2.4 GHz): 01 ~ 16
- Terminal IDs (2.4 GHz): 01 ~ 99

5.18.3 RF Terminal Properties

Below are the RF properties on terminals.

➢ 2.4 GHz Terminal Properties

Channel:	1 ~ 6
ID:	1 ~ 99
Time out:	1 ~ 99 seconds (duration of retries for sending data)
Output power:	1 level (64 mW)
Auto search:	0 ~ 99 seconds (automatically search for available channel when the connection to the current channel is lost)

CHECK_RF_BASE

Purpose To check if the terminal is connected to a base.

Syntax *A%* = CHECK_RF_BASE

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
0	No base found.
1	Base is present.

Example
```
IF (CHECK_RF_BASE = 1) THEN
…
END IF
```

See Also CHECK_RF_SEND

CHECK_RF_SEND

Purpose To check if data has been sent successfully or not.

Syntax *A%* = CHECK_RF_SEND

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
0	Fail to send.
1	Sent successfully.

Example
```
RESEND:
WRIT_COM(2, A$)
IF (CHECK_RF_SEND = 0) THEN
   GOTO RESEND
END IF
```

See Also CHECK_RF_BASE

GET_RF_CHANNEL

Purpose To retrieve the channel number of the terminal.

Syntax *A%* = GET_RF_CHANNEL

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
1 ~ 4	Channels available for 433 MHz.
1 ~ 6	Channels available for 2.4 GHz.

Example channel% = GET_RF_CHANNEL

See Also GET_RF_ID, GET_RF_POWER, SEARCH_RF_CHANNEL, SET_RF_CHANNEL

GET_RF_ID

Purpose To retrieve the ID of the terminal.

Syntax *A%* = GET_RF_ID

Remarks *"A%" is an integer variable to be assigned to the result.*

Example *Id% = GET_RF_ID*

See Also GET_RF_CHANNEL, GET_RF_POWER, SET_RF_ID

GET_RF_POWER

Purpose To get the level of the RF output power (applicable to 433 MHz only).

Syntax *A%* = GET_RF_POWER

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
1	10 dbm
2	5 dbm
3	4 dbm
4	0 dbm
5	-5 dbm

Example level% = GET_RF_POWER

See Also GET_RF_CHANNEL, GET_RF_ID, SET_RF_POWER

SEARCH_RF_CHANNEL

Purpose To automatically search for available channels when not being connected to a base for a specific period of time

Syntax SEARCH_RF_CHANNEL(*N%*)

Remarks "*N%*" is an integer variable, indicating a specified period of time in units of 1 second.

Example SEARCH_RF_CHANNEL(10)

See Also GET_RF_CHANNEL, SET_RF_CHANNEL, SET_RF_TIMEOUT

SET_RF_CHANNEL

Purpose To set the channel of the terminal.

Syntax SET_RF_CHANNEL(*N%*)

Remarks "N%" is an integer variable, indicating the specified channel for the terminal.

Value	Meaning
1 ~ 4	Channels available for 433 MHz.
1 ~ 6	Channels available for 2.4 GHz.

Example SET_RF_CHANNEL(1)

See Also GET_RF_CHANNEL, SEARCH_RF_CHANNEL, SET_RF_ID, SET_RF_POWER

SET_RF_ID

Purpose To set the ID of the terminal.

Syntax SET_RF_ID(*N%*)

Remarks "*N%*" is an integer variable, indicating the ID of the terminal.

Value	Meaning
1 ~ 45	IDs available for 433 MHz.
1 ~ 99	IDs available for 2.4 GHz.

Example SET_RF_ID(1)

See Also GET_RF_ID, SET_RF_CHANNEL, SET_RF_POWER

SET_RF_POWER

Purpose To set the RF output power.

Syntax SET_RF_POWER(*N%*)

Remarks "*N%*" is an integer variable, indicating the power level of the terminal.

Value	Meaning
1	10 dbm
2	5 dbm
3	4 dbm
4	0 dbm
5	-5 dbm

Example SET_RF_POWER(2)

See Also GET_RF_POWER, SET_RF_CHANNEL, SET_RF_ID

SET_RF_TIMEOUT

Purpose To set the duration of retries in seconds, for sending data.

Syntax SET_RF_TIMEOUT(*N%*)

Remarks "*N%*" is an integer variable, indicating a specified period of time in units of 1 second.

If set to 0, it will be determined by the system.

Example SET_RF_TIMEOUT(5)

See Also SEARCH_RF_CHANNEL

RF Practice

➢ Programming Flow

1. Set COM port (SET_COM_TYPE)

   Assign COM2 to RF communication, i.e. SET_COM_TYPE(2,5).

2. Open COM port (OPEN_COM)

   To initialize the RF module and set up connection, call OPEN_COM(2).

3. Set COM delimiter (COM_DELIMITER)

   For example, use carriage return (0x0d) as the string delimiter by calling COM_DELIMITER(2,13).

4. Data Transmission

   To transmit/receive data, call WRITE_COM(2) and READ_COM$(2) respectively.

5. Close COM port (CLOSE_COM)

   To disable the RF module, call CLOSE_COM(2).

5.18.4 RF Base Properties

Below are the RF properties on base stations.

➢ 433 MHz Base Properties

Mode:	1 – standalone, 2 – slave, 3 – master
Channel:	1 ~ 4
ID:	01 ~ 16
Group:	1 ~ 3
Time out:	1 ~ 99 seconds (duration of retries for sending data)
Output power:	1 ~ 5 levels (10, 5, 4, 0, -5 dbm)
Baud rate:	115200, 57600, 38400, 19200, 9600

➢ 2.4 GHz Base Properties

Mode:	1 – standalone, 2 – slave, 3 – master
Channel:	1 ~ 6
ID:	01 ~ 16
Group:	1
Time out:	1 ~ 99 seconds (duration of retries for sending data)
Output power:	100 mW
Baud rate:	115200, 57600, 38400, 19200, 9600

Note: If more than two base stations are connected together, the one connected to the host computer needs to be set to master mode, while the rest to slave mode.

5.18.5 Programming RF Base - RS-232 Host Command Sets

Automatically Update Status: @AT	**RF Base**

Purpose To update information of bases and terminals automatically.

Note that there will be no return of @BSbbgc if only one base exists.

Syntax @AT\r

Return @BSbbgc…\r (bbgc may repeat)

bb: Base ID (01 ~ 16)

g: Group number (1 ~ 3)

c: Channel number (1 ~ 4)

@TMbbtt…\r (tt may repeat)

bb: Base ID (01 ~ 16)

tt: Terminal ID (01 ~ 45)

Broadcast Data: @BC	**RF Base**

Purpose To broadcast data to all registered bases.

Note that all bases are connected via RS-485 as a daisy chain; therefore, data may not be sent to all registered bases if any of them fails.

Syntax @BC…ddd…\r

Return None

List Base Station @BS	**RF Base**

Purpose To get information of bases, such as their groups and channels.

Syntax @BS\r

Return @BSbbgc…\r (bbgc may repeat)

bb: Base ID (01 ~ 16)

g: Group number (1 ~ 3)

c: Channel number (1 ~ 4)

Set / Get Channel: @CH	RF Base

Purpose To set or get the Base's channel.

Syntax @CHbbc\r

bb: Base ID (01 ~ 16)

c: New channel (1 ~ 4); set 0 to get the Base's channel.

Return @CHbbCoCn\r

bb: Base ID (01 ~ 16)

Co: Original channel (1 ~ 4)

Cn: New channel (1 ~ 4)

Send Data to Terminal: @DT	RF Base

Purpose To send data to the terminal.

Syntax @DTttdd…\r

tt: Terminal ID (01 ~ 45)

dd: Data string

Return @Oktt\r (succeed)

@NGtt\r (fail)

@WTtt\r (busy; data will be sent later)

Receive Data from Terminal: @DT	RF Base

Purpose To receive data from the terminal.

Syntax @DTbbttdd…\r

(Received) bb: Base ID (01 ~ 16)

tt: Terminal ID (01 ~ 45)

dd: Data string

Command / Format Error: @ER	RF Base

Purpose If the command of data being sent to the Base is incorrect, the Base returns @ER\r.

Set / Get Group: @GP — RF Base (433 MHz only)

Purpose To set or get the Base's group.

Syntax @GPbbg\r

bb: Base ID (01 ~ 16)

g: New group (1 ~ 3); set 0 to get the Base's group.

Return @GPbbGoGn\r

bb: Base ID (01 ~ 16)

Go: Original group (1 ~ 3)

Gn: New group (1 ~ 3)

Header of Data Packets: @HD — RF Base

Purpose To enable or disable the header of a data packet. By default, the data received will be prefixed with "@DTbbtt", where bb refers to Base ID and tt refers to Terminal ID. If the header is disabled, there will be no prefix at all.

Syntax @HDbbc\r

bb: Base ID (01 ~ 16)

c: set 0 to disable the header; set 1 to enable the header (default: enable)

Return @HDbbCoCn\r

bb: Base ID (01 ~ 16)

Co: Original setting (1~ 4)

Cn: New setting (1 ~ 4)

Change ID: @ID — RF Base

Purpose To change the Base ID.

Syntax @IDbbBB\r

bb: Original Base ID (01 ~ 16)

BB: New Base ID (01 ~ 16)

Return @IDbbBB\r

bb: Original Base ID (01 ~ 16)

BB: New Base ID (01 ~ 16)

Set / Get Mode: @ME — RF Base

Purpose To set or get the Base's mode.

Syntax @MEbbm\r

bb: Base ID (01 ~ 16)

m: Base mode (1 ~ 3); set 0 to get the Base's mode.

Set Base Mode	
1	Standalone
2	Slave
3	Master

Return @MEbbmomn\r

bb: Base ID (01 ~ 16)

mo: Original mode (1 ~ 3)

mn: New mode (1 ~ 3)

Change Output Power: @PW — RF Base (433 MHz only)

Purpose To change the Base's output power.

Syntax @PWbbp\r

bb: Base ID (01 ~ 16)

p: Power level (1 ~ 5); set 0 to get the Base's output power.

Power Level	
1	10 dbm
2	5 dbm
3	4 dbm
4	0 dbm
5	-5 dbm

Return @PWbbPoPn\r

bb: Base ID (01 ~ 16)

Po: Original power level (1 ~ 5)

Pn: New power level (1 ~ 5)

Set Base Quiet: @QT — RF Base

Purpose To enable or disable the message ("@Oktt\r", "@NGtt\r", "@WTtt\r") generation of the Base when sending data to the terminal.

Syntax @QTbbs\r

bb: Base ID (01 ~ 16)

s: set 0 for responsive; set 1 for quiet (i.e. no message)

Return @QTbbSoSn\r

bb: Base ID (01 ~ 16)

So: Original state

Sn: New state

RS-232 Baud Rate: @SP — RF Base

Purpose To change the RS-232 baud rate setting of the Base.

Note that the connection has to be restarted with new setting.

Syntax @SPbbs\r

bb: Base ID (01 ~ 16)

s: Baud rate setting

Baud Rate Setting (bps)
115200
57600
38400
19200
9600
4800
2400
1200

Return None

List Terminal: @TM — RF Base

Purpose To list terminals that are registered to the Base or bases.

Syntax @TMbb\r

bb: Base ID (01 ~ 16); set 00 for all bases.

Return @TMbbtt…\r (tt repeat)

bb: Base ID (01 ~ 16)

tt: Terminal ID (01 ~ 45)

Transmission Timeout: @TO — RF Base

Purpose To set or get the timeout setting.

Syntax @TObbtt\r

bb: Base ID (01 ~ 16)

tt: Timeout setting (01 ~ 99 sec); set 00 to get the current timeout.

Return @TObbttTT\r

bb: Base ID (01 ~ 16)

tt: Original timeout (01 ~ 99 sec)

TT: New timeout (01 ~ 99 sec)

Update Program: @UP — RF Base

Purpose To enter the download mode for updating the Base program.

Syntax @UPbb\r

bb: Base ID (01 ~ 16)

Return @UPbb\r

bb: Base ID (01 ~ 16)

The Base is ready for downloading new program after returning this message.

5.19 Wireless Communications

Please refer to the previous section for port mapping.

5.19.1 Wireless Models

This section describes the commands relating to wireless network configuration. These command sets are only applicable to the models below.

Product Family	Optimus S (SP5500)	Optimus R (SP5600)
Wireless, optional	Bluetooth	IEEE 802.11b

5.19.2 Network Configuration

- **Net Configuration**

Network Configuration			Index		Configuration Item	Default Setup String
802.11b	SPP	PAN (BNEP)	GET_NET_ PARAMETER	SET_NET_ PARAMETER		
✓		✓	0 ~ 3		Remote IP	
✓		✓	-1	1	Local IP	
✓		✓	-2	2	Subnet Mask	
✓		✓	-3	3	Router IP	
✓		✓	-4	4	DNS Server IP	
✓	✓	✓	-5	5	Local Name	S/N
✓			-6	6	SS ID	
✓			-7 to -10	7 to 10	WEP Key 1 ~ 4	
✓		✓	-11	11	DHCP Enable	Enable
✓			-12	12	Authentication	Open
✓			-13	13	WEP Key Length	128 bits
✓			-14	14	System Scale	Medium
✓			-15	15	Default WEP Key	1
✓		✓	-16	16	Domain Name	
✓			-17	17	WEP Key Enable	Disable

✓			-18	18	EAP Enable	Disable
✓			-19	19	EAP ID	
✓			-20	20	EAP Password	
✓			-21	21	Power Saving Enable	Enable
✓			-22	22	Preamble Type	Long
✓			-23		MAC ID	
	✓	✓	-24		BT MAC ID	
	✓	✓	-25	25	BT Remote Name	
	✓	✓	-26	26	BT Security	Disable
	✓	✓	-27	27	BT PIN Code	
	✓	✓	-28	28	BT Broadcast	Enable
	✓	✓	-29	29	BT Power Saving	Enable
✓			-30	30	Ad-hoc	
	✓	✓	-40 to -47	40 to 47	BT Frequent Device List	
GSM/GPRS			**Index**		**Configuration Item**	**Default Setup String**
			GET_NET_ PARAMETER	**SET_NET_ PARAMETER**		
			-60		GSM Service Center	
			-61	61	GSM/GPRS PIN Code	
			-62	62	GPRS AP	
			-63		GSM Net	
			-64	64	GSM Modem Dial Number	

GET_NET_PARAMETER$

Purpose To get network settings.

Syntax *A$* = GET_NET_PARAMETER$(*index%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*index%*" is an integer variable, indicating a specific configuration item by index number.

Error code:

Value	Meaning
0	Normal status: the connection is open.
3000	Invalid index number.
3004	Connection is closed.
3012	START TCPIP not applied.

Example NetSetting$ = GET_NET_PARAMETER$(0)

See Also SET_NET_PARAMETER, SOCKET_IP$, TCP_ERR_CODE

SET_NET_PARAMETER

Purpose To configure network settings.

Syntax SET_NET_PARAMETER(*index%, A$*)

Remarks Note that it is not necessary to configure the setting every time.

"*index%*" is an integer variable, indicating a specific configuration item by index number.

"*A$*" is a string variable indicating the network setting to be configured.

Configuration Item	Setup string
(Boolean type)	"Enable" / "Disable"
Authentication	"Open" / "Share"
WEP Key Length	"64 bits" / "128 bits"
System Scale	"Low" / "Medium" / "High"
Preamble Type	"Short" / "Long" / "Both"

Example

```
SET_NET_PARAMETER (1, "192.168.1.241")  ' set local IP
SET_NET_PARAMETER (11, "Disable")       ' disable DHCP
SET_NET_PARAMETER (12, "Share")         ' set authentication "Share Key"
SET_NET_PARAMETER (22, "Short")         ' set preamble type "Short"
```

See Also GET_NET_PARAMETER$, IP_CFG, TCP_ERR_CODE

➢ Net Status

Network Configuration			Index	Configuration Item	Return Value	
802.11b	**SPP**	**PAN (BNEP)**	**GET_NET_ STATUS**			
✓			1	Connection status	0 1	Disabled Connected
✓			2	Communication quality	0 to 10 10 to 15 15 to 30 30 to 50 50 to 80	Very poor Poor Fair Good Very good
✓			3	Signal strength	0 to 30 30 to 60 60 to 120	Weak Moderate Strong
✓			4	Noise level	1 2 to 3 4 to 5	Weak Moderate Strong
✓			5	Channel number	1 to 11	
✓			6	Transmission rate	1 2 4 8	1 Mbps 2 Mbps 5.5 Mbps 11 Mbps
✓		✓	7	Terminal IP	0 1	Not ready Ready
	✓	✓	8	Bluetooth status	0 1	Disabled Connected
	✓	✓	9	Bluetooth RSSI	-10 to -6 -6 to 5 5 to 30	Weak Moderate Strong
GSM/GPRS			**Index**	**Configuration Item**	**Return Value**	
			GET_NET_ STATUS			
			10	Connection status	0 1	Disabled Connected

	11	GSM RSSI	0	-113 dbm or less
			1	-111 dbm
			2	-109 dbm
			... (3 ~ 29)	... (1 for +2 dbm)
			30	-53 dbm
			31	-51 dbm or greater
			99	Not known or not detectable
	12	GSM/GPRS PIN Code	0	Disabled
			1	PIN code required

GET_NET_STATUS

Purpose To get network status.

Syntax *A%* = GET_NET_STATUS(*index%*)

Remarks Note that it is necessary to define the DNS server IP before executing this command.

"*A%*" is an integer variable to be assigned to the result.

"*index%*" is an integer variable indicating a specific configuration item by index number.

Example
```
nQuality = GET_NET_STATUS(2)          ' check communication quality
```

See Also GET_WLAN_STATUS, TCP_ERR_CODE

GET_WLAN_STATUS

Purpose To get network status..

Syntax *A%* = GET_WLAN_STATUS(*index%*)

Remarks **This command is to be replaced by GET_NET_STATUS.**

Example
```
nQuality = GET_WLAN_STATUS (2)     ' check communication quality
```

See Also GET_NET_STATUS, TCP_ERR_CODE

5.19.3 IEEE 802.11b

IEEE 802.11b is an industrial standard for Wireless Local Area Networking (WLAN), which enables wireless communications over a long distance.

The speed of the connection between wireless devices will vary with range and signal quality. To maintain a reliable connection, the 802.11b system automatically fallback from 11 Mbps to 5.5, 2 or 1 Mbps as range increases or signal quality decreases.

Frequency Range:	2.4 ~ 2.5 GHz
Data Rate:	11, 5.5, 2, 1 Mbps auto-fallback
Connected Devices:	1 for Peer-to-Peer mode (No AP)
	Multiple for Server-to-Client mode (AP required)
Protocol:	IP/TCP/UDP
Coverage:	250 meters line-of-sight
Maximum Output Power:	100 mW
Spread Spectrum:	DSSS
Modulation:	DBPSK (1 Mbps), DQPSK (2 Mbps), CCK (11 Mbps)
Standard:	IEEE 802.11b, interoperable with Wi-Fi devices

Note: All specifications are subject to change without prior notice.

TCP/IP Networking

Here are the BASIC functions and statements relating to TCP/IP networking.

Commands for triggering the TCPIP event: *OFF TCPIP, ON TCPIP GOSUB...*

DNS_RESOLVER

Purpose To get the IP address of the remote device.

Syntax *IP$* = DNS_RESOLVER(*A$*)

Remarks Note that it is necessary to define the DNS server IP before executing this command.

"*IP$*" is a string variable to be assigned to the result.

"*A$*" is a string variable, indicating a specific domain name.

Example GetIP$ = DNS_RESOLVER ("www.yahoo.com")

See Also TCP_ERR_CODE

GET_TCPIP_MESSAGE

Purpose To get the message of TCPIP event trigger.

Syntax *A%* = GET_TCPIP_MESSAGE

Remarks This command can also be called in normal program to detect the TCP/IP status by polling method. Once it is fetched, the message will be cleared by the system.

When entering TCPIP event trigger, the first thing is to call this routine so that the trigger message will be cleared out.

"*A%*" is an integer variable to be assigned to the result.

Value	Meaning
4000	Connection #0 overflow
4001	Connection #1 overflow
4002	Connection #2 overflow
4003	Connection #3 overflow
4014	Networking initialization error
4015	Port initialization error

4020	Connection #0 is connected
4021	Connection #1 is connected
4022	Connection #2 is connected
4023	Connection #3 is connected
4040	Connection #0 is disconnected
4041	Connection #1 is disconnected
4042	Connection #2 is disconnected
4043	Connection #3 is disconnected
4060	Connection #0: data is coming
4061	Connection #1: data is coming
4062	Connection #2: data is coming
4063	Connection #3: data is coming
4080	IP is ready

Example
```
ON TCPIP GOSUB TCPIP_Trigger
...
TCPIP_Trigger:
MSG% = GET_TCPIP_MESSAGE
...
```

See Also OFF TCPIP, ON TCPIP GOSUB…, TCP_OPEN

IP_CFG or IP_CONFIGURE

Purpose To configure the TCP/IP setting.

Syntax IP_CFG(*index%, IP$*) or IP_CONFIGURE(*index%, IP$*)

Remarks **This command is to be replaced by SET_NET_PARAMETER.**

Note that it is not necessary to configure the setting every time.

"*index%*" is an integer variable, indicating a specific configuration item.

"*IP$*" is a string variable indicating the IP address that is to be configured.

Example
```
IP_CFG (1, "192.168.1.241")          ' set local IP
IP_CFG (2, "255.255.255.0")          ' set IP of subnet mask
IP_CFG (3, "192.168.1.250")          ' set IP of default gateway
IP_CFG (4, "168.95.1.1")             ' set IP of DNS server
```

See Also SET_NET_PARAMETER, TCP_ERR_CODE

NCLOSE

Purpose To close a TCP/IP connection.

Syntax NCLOSE(*N%*)

Remarks "*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

Example NCLOSE (0)

See Also NREAD$, NWRITE, TCP_ERR_CODE, TCP_OPEN

NREAD$

Purpose To read data via a TCP/IP connection.

Syntax *A$* = NREAD$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

Example A$ = NREAD$ (0)

See Also NCLOSE, NWRITE, OFF TCPIP, ON TCPIP GOSUB…, SOCKET_HAS_DATA, TCP_ERR_CODE, TCP_OPEN

NWRITE

Purpose To write data to a TCP/IP connection.

Syntax NWRITE (*N%, A$*)

Remarks "*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

"*A$*" is a string variable, representing the string to be sent to the connection.

Example NWRITE (0, "Hello")

See Also NCLOSE, NREAD$, SOCKET_CAN_SEND, TCP_ERR_CODE, TCP_OPEN

SOCKET_CAN_SEND

Purpose To check if the data can be sent out or not.

Syntax *A%* = SOCKET_CAN_SEND(*N%, L%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
0	Normal - data can be sent
3000	Invalid connection number
3004	Connection is closed
3007	Cannot send data
3012	Never run START TCPIP

"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

"*L%*" is an integer variable, indicating the length of data.

Example A% = SOCKET_CAN_SEND (0, 10)

See Also NCLOSE, NWRITE, TCP_ERR_CODE, TCP_OPEN

SOCKET_HAS_DATA

Purpose To check if the buffer of connection has data or not.

Syntax *A%* = SOCKET_HAS_DATA(*N%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
0	Normal - data in buffer
3000	Invalid connection number
3004	Connection is closed
3005	No data
3012	Never run START TCPIP

"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

Example A% = SOCKET_HAS_DATA(0)

See Also NCLOSE, NREAD$, OFF TCPIP, ON TCPIP GOSUB…, TCP_ERR_CODE, TCP_OPEN

SOCKET_IP$

Purpose To get network settings.

Syntax *A$* = SOCKET_IP$(*port%*)

Remarks **This command is to be replaced by GET_NET_PARAMETER$.**

Example NetSetting$ = SOCKET_IP$(0)

See Also GET_NET_PARAMETER$, TCP_ERR_CODE

SOCKET_OPEN

Purpose To check if the connection is open or not.

Syntax *A%* = SOCKET_OPEN(*N%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
0	Normal - connection is open
3000	Invalid connection number
3004	Connection is closed
3012	Never run START TCPIP

"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

Example ConnectState% = SOCKET_OPEN(0)

See Also TCP_ERR_CODE, TCP_OPEN

START TCPIP

Purpose To enable communication via TCP/IP.

Syntax START TCPIP

START TCPIP(N%)

Remarks This routine is used to perform general initialization. It must be the first network function call, and cannot be called again unless STOP TCPIP has been called.

- For the 8x60 series, START TCPIP is to enable its built-in Bluetooth module. It also supports START TCPIP(3) to activate the GPRS functionality on the mobile phone via its built-in Bluetooth.
- For the 8x70 series, START TCPIP is to enable its built-in 802.11b module.
- For the CPT-8500, START TCPIP or START TCPIP(0) is to enable its optional 802.11b module by default.

  To enable its built-in Bluetooth module, you have to use START TCPIP(1).

  To activate its own GPRS functionality, use START TCPIP(2).

  To activate the mobile's GPRS functionality, use START TCPIP(3).

"*N%*" is an integer variable, indicating which wireless module is to be used (all applicable to the CPT-8500, and partially to 8x60).

Value	Meaning
0	802.11b (default)
1	Bluetooth
2	GPRS
3	Mobile's GPRS via Bluetooth

Example START TCPIP ' this is hardware-dependent

See Also OFF TCPIP, ON TCPIP GOSUB…, STOP TCPIP, TCP_ERR_CODE, TCP_OPEN

STOP TCPIP

Purpose To disable communication via TCP/IP.

Syntax STOP TCPIP

Remarks

Example STOP TCPIP

See Also START TCPIP, TCP_OPEN

TCP_ERR_CODE

Purpose To check the result after executing any command relating to TCP/IP (except STOP TCPIP).

Syntax *A%* = TCP_ERR_CODE

Remarks "*A%*" is an integer variable to be assigned to the result, indicating an error description.

If a routine is working normally, the return value will be 0 in general. However, it will return "N%" (the connection number) for TCP_OPEN and "1 ~ 255" (the length of data being read) for NREAD$.

Value	Meaning
0	Normal*
3000	Invalid connection number
3001	Connection is already opened.

3002	Undefined local port in server mode
3003	Undefined remote port in client mode
3004	Connection is closed.
3005	No data received in buffer
3006	Data too long
3007	Networking is busy or data is too long.
3008	Data transmission error
3009	Hardware initialization failure
3010	START TCPIP has already been running. May need STOP TCPIP.
3011	All connections are unavailable.
3012	Never run START TCPIP
-10	Parameter error
-11	Host is not reachable.
-12	Time out
-13	Hardware failure
-14	Protocol error
-15	No buffer space
-16	Invalid connection block
-17	Invalid pointer argument
-18	Operation would block.
-19	Message too long
-20	Protocol unavailable
-30	Unknown remote name
-31	DNS protocol error (package class)
-32	DNS protocol error (package type)
-33	Remote name too long (more than 38 characters)

Example ERR% = TCP_ERR_CODE

See Also DNS_RESOLVER, GET_NET_STATUS, GET_WLAN_STATUS, IP_CFG, NCLOSE, NREAD$, NWRITE, OFF TCPIP, ON TCPIP GOSUB…, SOCKET_CAN_SEND, SOCKET_HAS_DATA, SOCKET_IP$, SOCKET_OPEN, START TCPIP, TCP_OPEN, SET_NET_PARAMETER

TCP_OPEN

Purpose To open a connection via TCP/IP protocol.

Syntax TCP_OPEN(*N%, IP$, RP%, LP%,* {*Protocol%*}, {*Delimiter%*})

Remarks Note that the function must be called before using any socket read/write commands.

"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

"*IP$*" is a string variable, indicating the IP address of the remote port. If it is set to "0.0.0.0", the connection will become server mode and the LP% must be defined.

"*RP%*" is an integer variable, indicating the port number of the remote port, which is to be connected. If it is 0 (=none), any remote port is available. However, it has to be set to 0 when in server mode.

"*LP%*" is an integer variable, indicating the port number of the local port. If it is 0 (=none), the port number will be defined by the system. However, it has to be set to 0 when in client mode.

	Server mode	**Client mode**
N%	0 ~ 3	0 ~ 3
IP$	"0.0.0.0"	Required
RP%	0	Required
LP%	Required	0

"*Protocol%*" is an integer variable, indicating the networking protocol in use. This parameter is optional and it is set to 0 by default (using TCP/IP protocol). If it is set to 1, the system will use UDP/IP protocol.

"*Delimiter%*" is an integer variable, indicating whether to transmit the delimiter or not. This parameter is optional and it is set to 0x0d (Carriage Return) by default. The valid values range from 0 to 255. If it is set to -1, the system will not transmit any delimiter.

Example TCP_OPEN (0, "0.0.0.0", 0, 23)

TCP_OPEN (1, "0.0.0.0", 0, 24)

TCP_OPEN (2, "0.0.0.0", 0, 25, 1)

TCP_OPEN (3, "0.0.0.0", 0, 26, 0, 59)

See Also GET_TCPIP_MESSAGE, NCLOSE, NREAD$, NWRITE, OFF TCPIP, ON TCPIP GOSUB…, SOCKET_CAN_SEND, SOCKET_HAS_DATA, SOCKET_OPEN, START TCPIP, STOP TCPIP, TCP_ERR_CODE

TCP/IP Practice

➢ TCP/IP Configuration

The parameters can be accessed via GET_NET_PARAMETER$ and SET_NET_PARAMETER.

➢ Programming Flow

1. Set networking variables (SET_NET_PARAMETER)

2. Initialize the 802.11b/Bluetooth module and networking protocol stack

   (1) START TCPIP

   The 802.11b/Bluetooth module will not be powered until the START TCPIP routine is called. When the application program stops using the network, the STOP TCPIP routine can be called to terminate networking and shut down the 802.11b/Bluetooth module so that it can save power. Note that any previous network connection and data will be lost after calling STOP TCPIP. To enable the networking again, it is necessary to call START TCPIP again.

   (2) GET_TCPIP_MESSAGE

   The START TCPIP routine does the first stage of the initialization process, and it will generate a system task to finish the rest of the process. When the START TCPIP routine returns, the initialization process might not have been done yet. Therefore, it is necessary for the application program to check whether the status is "IP is ready" by calling GET_TCPIP_MESSAGE or GET_NET_STATUS before it proceeds to perform any networking operations.

   (3) GET_NET_STATUS

   Once the initialization process is done, the status of the 802.11b/Bluetooth module can be retrieved from the system. It will be periodically updated by the system. The application program must explicitly call the GET_NET_STATUS routine to get the latest status.

3. Open a Connection

   (1) TCP_OPEN

   Before reading and writing to the remote host, a connection must be established (opened). The TCP_OPEN routine is used to open a connection. The application program needs to define a connection number (0~3), so that it can identify a particular connection in subsequent calls to other TCP/IP stack routines.

   (2) GET_TCPIP_MESSAGE

   It is necessary for the application to check whether the status of the particular connection is "connected" by calling GET_TCPIP_MESSAGE before it proceeds to perform any read/write operations.

4. Data transmission

   (1) SOCKET_CAN_SEND

   Before sending data to the network, SOCKET_CAN_SEND can be used to check if there is enough buffer size to write out the data immediately. It also can be used to check if the data being sent is more than 4 packets when there is no response from the remote host.

   (2) NREAD$/NWRITE

   The routines NREAD$ and NWRITE are used to send and receive data on the network respectively.

5. Other useful functions

   There are other routines for obtaining additional information or setting control for a connection.

   (1) SOCKET_OPEN, SOCKET_HAS_DATA

   To check the connection status by polling method.

   (2) SOCKET_IP$

   To get the networking configuration and the remote site IP address.

   (3) TCPIP_ERR_CODE

   To get the operation result after calling any TCPIP routines.

   (4) TCPIP event trigger:

   ON TCPIP GOSUB… and OFF TCPIP are used to get higher working performance. Once the TCPIP event occurs, it is necessary for the application program to check the trigger type by getting the value of the GET_TCPIP_MESSAGE routine.

6. Close a Connection

   The NCLOSE routine is used to terminate a particular connection when the application program does not use it any more.

5.19.4 Bluetooth

- Serial Port Profile (SPP) is for ad hoc networking, without going through any access point.
- Personal Area Networking Profile (PAN) makes use of Bluetooth Network Encapsulation Protocol (BNEP) for IP networking over Bluetooth. Access points (AP) are required. Use the same commands as for 802.11b - TCP/IP networking.
- Dial-Up Networking Profile (DUN) makes use of a Bluetooth modem or mobile phone as a wireless modem.

Frequency Range:	2.4 ~ 2.5 GHz
Data Rate:	721 kbps
Connected Devices:	1 for SPP/DUN mode
	Multiple for PAN mode (Server-to-Client, AP required)
Profiles:	SPP, PAN, DUN
Coverage:	250 meters line-of-sight
Maximum Output Power:	100 mW
Spread Spectrum:	FHSS
Modulation:	GFSK
Standard:	Bluetooth version 1.2

Note: All specifications are subject to change without prior notice.

For the 8x60 series, it makes use of the pairing procedure to keep record of the latest connected device(s) for different modes, regardless of authentication enabled or not. This is so-called “Frequent Device List”.

Coverage of Frequent Device List	
SPP mode	Only one device is listed for quick connection.
DUN mode	Only one device is listed for quick connection.
PAN mode	Up to eight access points are listed for roaming purpose.

Once the pairing procedure is completed and the list is generated successfully, next time the terminal will automatically connect to the listed device(s) without going through the pairing procedure.

To complete the pairing procedure, it consists of two steps: (1) to discover the Bluetooth devices in range, and (2) to page one of them that provides a particular service. These are handled by BT_INQUIRY$ and BT_PAIRING respectively.

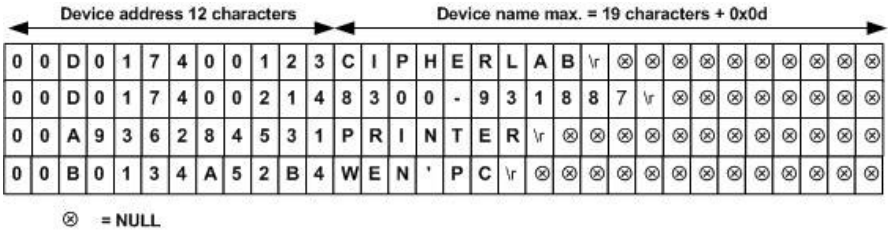
BT_INQUIRY$

Purpose To search any available Bluetooth devices in range.

Syntax *A$* = BT_INQUIRY$

Remarks It takes about 20 seconds to get the Bluetooth device information of whomever in range. The string contains address (12 bytes) and name (20 bytes) of the devices. Note that there might be many devices concatenated together, each occupying 32 bytes.

Information regarding the Bluetooth devices in range will be put in the format of MENU() string as shown below:


Device address 12 characters
Device name max. = 19 characters + 0x0d
0 0 D 0 1 7 4 0 0 1 2 3 C I P H E R L A B \r ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗
0 0 D 0 1 7 4 0 0 2 1 4 8 3 0 0 - 9 3 1 8 8 7 \r ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗
0 0 A 9 3 6 2 8 4 5 3 1 P R I N T E R \r ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗
0 0 B 0 1 3 4 A 5 2 B 4 W E N ' P C \r ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗
⊗ = NULL

Example …

```
MENU_STR$ = BT_INQUIRY$
I% = MENU(MENU_STR$)
…
```

See Also BT_PAIRING

BT_PAIRING

Purpose To check if the device can provide a specific type of service, and (if required), the PIN code for authentication is matching.

Syntax *A%* = BT_PAIRING(*addr$, type%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
1	Pairing successfully
0	Service unavailable or wrong PIN code

"*addr$*" is a string variable, indicating the IP address of the device.

"*type%*" is an integer variable, indicating a specific type of service.

Value (type%)	Meaning
1	PAN (AP required)
3	SPP
4	DUN

It will try to pair with any Bluetooth device that has the specific type of service. If authentication is enabled, then correct PIN code will be required for setting up the Link Key. Once the pairing procedure is completed, the MAC ID of the remote device will be recorded in the “Frequent Device List” for quick connection in the future.

Example

```
…
MENU_STR$ = BT_INQUIRY$
I% = MENU(MENU_STR$)
DEVICE$ = MID$(MENU_STR$, 1+32*(I%-1), 32)
R% = BT_PAIRING(DEVICE$, 3)
…
```

See Also BT_INQUIRY$

Frequent Device List

➢ Getting Frequent Device List

The length of Frequent Device List by calling GET_NET_PARAMETE$ is 83 characters:

```
LIST$ = GET_NET_PARAMETER$(40)
```

The first character of Frequent Device List is the service type that the device is engaged. Currently, there are three types that have been defined:

Service Type		In Frequent Device List
1	PAN (= AP mode, i.e. access points are required.)	Max. 8 devices are recorded
3	SPP	Only 1 device is recorded
4	DUN	Only 1 device is recorded

Note: If bit 7 = 1, it means that this device is currently connected.

After the service type, from the second to the 13th character stands for the string of MAC ID.

The next property after MAC ID is Device Name, which consists of up to 20 characters and ends with a delimiter code "\r".

The next property after Device Name is PIN code, which consists of up to 17 characters and ends with a delimiter code "\r".

The last property of Frequent Device List is Link Key, which is normally generated when the pairing procedure is completed. This unique Link Key is applied to the specific device connection only. Once the connection is renewed with a different device, a new Link Key will be generated.

Length	Property	Char												
1	**Service Type**	1	132											
2 ~ 13	**MAC ID**	12	"0"	"0"	"D"	"0"	"1"	"7"	"3"	"0"	"1"	"2"	"3"	"4"
14 ~ 33	**Device Name**	20	"M"	"y"	" "	"N"	"A"	"M"	"E"	\r	0	……...		0
34 ~ 50	**PIN Code**	17	"1"	"2"	"3"	"4"	\r	0	…….					0
51 ~83	**Link Key**	33	"1"	"2"	"4"	"F"	"5"	"3"	…….					\r

Note: Make sure to put "\r" as a delimiter for Device Name, PIN Code, and Link Key.

Example code:

```
FREQ_DEV$ = “”
CLS
FOR K% = 1 TO 8
    FDL$ = “”
    FDL$ = GET_NET_PARAMETER$(-39-K%)
        IF MID$(FDL$,1,1)<>CHR$(0) THEN
            DEV$ = MID$(FDL$,14,20)
            MAC_ID$ = MID$(FDL$,2,12)
            MACHINE$ = MID$(FDL$,1,1)
            FREQ_DEV$ = FREQ_DEV$+DEV$
            FREQ_MAC$ = FREQ_MAC$+MAC_ID$
            FREQ_MC$ = FREQ_MC$+MACHINE$
        END IF
NEXT K%
I% = MENU(FREQ_DEV$)
```

➢ Setting Frequent Device List

To enable quick connection to a specific device without going through the inquiry and pairing procedure, a user-definable Frequent Device List can be set up by calling SET_NET_PARAMETER. If there is an existing Frequent Device List generated from the inquiry and pairing procedure, it then may be partially or overall updated by this, and vice versa.

There are five fields: Service Type, MAC ID, Device Name, PIN Code, and Link Key. If authentication is disabled, you only need to specify the first three fields. Otherwise, the PIN code field needs to be specified for generating Link Key.

Example code (1):

```
‘setting up a DUN Frequent Device List without authentication by
calling SET_NET_PARAMETER:
…
FDL$ = CHR$(4+128)+”00d017401234”+”TestDev.”+CHR$(13)
SET_NET_PARAMETER(40,FDL$)
```

…

Example code (2):

```
‘setting up a SPP Frequent Device List with authentication (needs PIN
code) by calling SET_NET_PARAMETER:

…

FDL$ = CHR$(3+128) + ”00d017401234” + ”TestDev.” + CHR$(13) + ”1234”
+CHR$(13)

…

SET_NET_PARAMETER(40,FDL$)
```

Bluetooth Practice

➢ Bluetooth Configuration

The parameters can be accessed via GET_NET_PARAMETER$ and SET_NET_PARAMETER.

➢ SPP Application

1. Set COM port (SET_COM_TYPE)

   Assign either COM1 or COM2 to Bluetooth communication, e.g. SET_COM_TYPE(2,5)

2. Configure COM port (SET_COM)

   SET_COM(N%, Baudrate%, Parity%, Data%, Handshake%) – To determine which SPP mode to initialize, use the second parameter regarding baud rate setting.

   To initialize SPP Master mode, call SET_COM(2,4,1,1,1).

   To initialize SPP Slave mode, call SET_COM(2,1,1,1,1).

Parameters	Values	Remarks
#1: N%	2	Indicates Bluetooth COM port is to be set.
#2: (Baudrate%)	1: 115200 bps 2: 76800 bps 3: 57600 bps 4: 38400 bps 5: 19200 bps 6: 9600 bps 7: 4800 bps 8: 2400 bps	The baud rate setting is NOT applicable to Bluetooth. Simply assign 1 for SPP Slave; 4 for SPP Master; 5 for DUN.
#3: (Parity%)	1: None 2: Odd Parity 3: Even Parity	The parity setting is NOT applicable to Bluetooth. Simply assign 1 for Bluetooth.
#4: (Data%)	1: 7 data bits 2: 8 data bits	The data bits setting is NOT applicable to Bluetooth. Simply assign 1 for Bluetooth.
#5: (Handshake%)	1: None 2: CTS/RTS 3: XON/XOFF 4: Wedge Emulator	The handshake setting is NOT applicable to Bluetooth. Simply assign 1 for Bluetooth SPP/DUN, or, assign 4 for Bluetooth Wedge Emulator.

3. Open COM port (OPEN_COM)

   To initialize the Bluetooth module and set up connection, call OPEN_COM(2).

4. Detect Connection

   To detect if the connection is completed, call GET_NET_STATUS(8). For example:

```
LOOP003:
IF GET_NET_STATUS(8) = 0 THEN GOO LOOP003
BEEP(4400,4)
CLS
PRINT "Connect OK"
```

5. Data Transmission

   To transmit/receive data, call WRITE_COM(2) and READ_COM$(2) respectively.

6. Detect Connection

   To detect if the connection is maintained, call GET_NET_STATUS(8). For example:

```
IF GET_NET_STATUS(8) = 0 THEN
BEEP(3300,4)
CLOSE_COM(2)
END IF
```

7. Close COM port (CLOSE_COM)

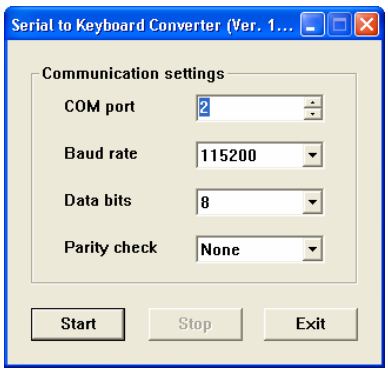
   To disable the Bluetooth module, call CLOSE_COM(2).

➢ PAN Application

All the program codes are compatible with TCP/IP Networking.

➢ Keyboard Emulator

The program "Serial to Keyboard Converter" is developed by 3rd party. It is a utility to let users convert Bluetooth SPP data to keyboard input.



The “Serial to Keyboard Converter” helps to develop a keyboard key in application without any serial port input function. It supports multi regions, i.e., an application can make use of this tool for varying keyboard layout.

The 8x60 series allows application send data for the purpose of keyboard input via Bluetooth SPP in general wedge functions, such as SEND_WEDGE, SET_WEDGE, and WEDGE_READY. That is, users can upgrade a normal wedge application to a “wireless” wedge application. All you need to do is to modify a few codes as shown below.

SET_COM(N%, Baudrate%, Parity%, Data%, Handshake%) – To set the wedge emulation flag, use the last parameter regarding hardware handshake setting.

```
SET_COM_TYPE(2,5)
SET_COM(2,1,1,1,4)
OPEN_COM(2)
```

And then, use the normal wedge functions to send data.

```
SET_COM_TYPE(2,5)
SET_COM(2,1,1,1,4)
OPEN_COM(2)
CLS
PRINT “Wait to Connect”
LOOP000:
IF WEDGE_READY = 0 THEN GOTO LOOP000
BEEP(4400,4)
CLS PRINT “OK! Try to Send”

LOOP:
KeyData$ = INKEY$
IF KeyData$ = “” THEN GOTO LOOP

IF KeyData$ = “0” THEN
    IF WEDGE_READY = 1 THEN
        PRINT “READY”
    ELSE
        PRINT “NOT READY”
    END IF
ELSE IF KeyData$ = “1” THEN
    SEND_WEDGE(“Hello”)
ELSE IF KeyData$ = “2” THEN
    PRINT “Hello”
END IF
GOTO Loop
```

5.20 File Manipulation

This section describes the commands relating to file manipulation. There are two different types of file structures supported in BASIC.

- DAT Files
- DBF File and IDX Files

For portable terminals, a BASIC program can have up to 6 transaction files and 5 DBF files (3 IDX files for each DBF file).

5.20.1 DAT Files

This one has a sequential file structure, which is much like the ordinary sequential file but is modified to support FIFO structure. We call this type of file as DAT file. Because DAT files are usually used to store transaction data, they are also referred to as Transaction files. The length of each record in the transaction file is limited to 250 bytes. For portable terminals, a BASIC program can have up to 6 transaction files.

DEL_TRANSACTION_DATA

Purpose To remove a block of transaction data from the default transaction file.

Syntax DEL_TRANSACTION_DATA(*N%*)

Remarks This command will only have an effect on the first (default) transaction file.

"*N%*" is an integer variable, determining how many transaction records to be deleted and how to delete.

If "*N%*" is a *positive* integer, the specified number of records will be deleted from the top of the transaction file 1, i.e. the oldest records will be deleted.

If "*N%*" is a *negative* integer, the specified number of records will be deleted from the bottom of the transaction file 1, i.e. the latest records will be deleted.

Example

```
...
PRINT "Discard the latest transaction? (Y/N)"
...
Loop:
  KeyData$ = INKEY$
  IF KeyData$ = "" THEN
    GOTO Loop
  ELSE IF KeyData$ = "Y" THEN
    DEL_TRANSACTION_DATA(-1)
  END IF
  ...
```

See Also DEL_TRANSACTION_DATA_EX, EMPTY_TRANSACTION

DEL_TRANSACTION_DATA_EX

Purpose To remove a block of transaction data from a specified transaction file.

Syntax DEL_TRANSACTION_DATA_EX(*file%, N%*)

Remarks "*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file the command is to affect. The command DEL_TRANSACTION_DATA_EX(1, *N%*) works the same as the command DEL_TRANSACTION_DATA(*N%*).

"*N%*" is an integer variable, determining how many transaction records to be deleted and how to delete.

If "*N%*" is a *positive* integer, the specified number of records will be deleted from the top of the transaction file 1, i.e. the oldest records will be deleted.

If "*N%*" is a *negative* integer, the specified number of records will be deleted from the bottom of the transaction file 1, i.e. the latest records will be deleted.

Example

```
...
PRINT "Discard the latest transaction? (Y/N)"
...
Loop:
  KeyData$ = INKEY$
  IF KeyData$ = "" THEN
    GOTO Loop
  ELSE IF KeyData$ = "Y" THEN
    DEL_TRANSACTION_DATA_EX(TransFile%, -1)
  END IF
  ...
```

See Also DEL_TRANSACTION_DATA, EMPTY_TRANSACTION_EX

EMPTY_TRANSACTION

Purpose To remove all the transaction data from the default transaction file.

Syntax EMPTY_TRANSACTION

Remarks This command will only have an effect on the first (default) transaction file.

Example

```
…
PRINT “Remove all the transaction data? (Y/N)”
…
Loop:
  KeyData$ = INKEY$
  IF KeyData$ = “” THEN
    GOTO Loop
```

```
ELSE IF KeyData$ = “Y” THEN
   EMPTY_TRANSACTION
END IF
```

See Also DEL_TRANSACTION_DATA, EMPTY_TRANSACTION_EX

EMPTY_TRANSACTION_EX

Purpose To remove all the transaction data from a specified transaction file.

Syntax EMPTY_TRANSACTION_EX(*file%*)

Remarks "*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file the command is to affect. The command EMPTY_TRANSACTION_EX(1) works the same as the command EMPTY_TRANSACTION.

Example EMPTY_TRANSACTION_EX(6)

See Also DEL_TRANSACTION_DATA_EX, EMPTY_TRANSACTION

GET_TRANSACTION_DATA$

Purpose To read a transaction record from the default transaction file.

Syntax *A$* = GET_TRANSACTION_DATA$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the transaction data.

"*N%*" is an integer variable, indicating the ordinal number of the record to be read from the first transaction file.

Example

```
...
WHILE (TRANSACTION_COUNT > 0)
   TransactionData$ = GET_TRANSACTION_DATA$(1)
   WRITE_COM(1, TransactionData$)
   DEL_TRANSACTION_DATA(1)
   WEND
```

See Also GET_TRANSACTION_DATA_EX$, SAVE_TRANSACTION, UPDATE_TRANSACTION

GET_TRANSACTION_DATA_EX$

Purpose To read a transaction record from a specified transaction file.

Syntax *A$* = GET_TRANSACTION_DATA_EX$(*file%*, *N%*)

Remarks "*A$*" is a string variable to be assigned to the transaction data.

"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. The command GET_TRANSACTION_DATA_EX$(1,1) works the same as the command GET_TRANSACTION_DATA$(1).

"*N%*" is an integer variable, indicating the ordinal number of the record to be read from the first transaction file.

Example

```
...
WHILE (TRANSACTION_COUNT > 0)
  TransactionData$ = GET_TRANSACTION_DATA_EX$(TransFile%,1)
  WRITE_COM(1, TransactionData$)
  DEL_TRANSACTION_DATA_EX(TransFile%,1)
  WEND
```

See Also GET_TRANSACTION_DATA$, SAVE_TRANSACTION_EX

UPDATE_TRANSACTION_EX

SAVE_TRANSACTION

Purpose To save (append) a transaction record to the default transaction file.

Syntax SAVE_TRANSACTION(*data$*)

Remarks "*data$*" is a string variable, representing the string to be saved in the first (default) transaction file.

Example

```
ON READER(1) GOSUB BcrData_1
  ...
BcrData_1:
  Data$ = GET_READER_DATA$(1)
  PRINT Data$
  SAVE_TRANSACTION(Data$)
  IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved."
  RETURN
```

See Also GET_TRANSACTION_DATA$, SAVE_TRANSACTION_EX, UPDATE_TRANSACTION

SAVE_TRANSACTION_EX

Purpose To save (append) a transaction record to a specified transaction file.

Syntax SAVE_TRANSACTION_EX(*file%*, *data$*)

Remarks "*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. The command SAVE_TRANSACTION_EX(1,data$) works the same as the command SAVE_TRANSACTION(data$).

"*data$*" is a string variable, representing the string to be saved in the specified transaction file.

Example

```
ON READER(1) GOSUB BcrData_1
...
BcrData_1:
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  PRINT Data$
  SAVE_TRANSACTION_EX(TransFile%,Data$)
  IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved."
  RETURN
```

See Also GET_TRANSACTION_DATA_EX$, SAVE_TRANSACTION, UPDATE_TRANSACTION_EX

TRANSACTION_COUNT

Purpose To get the total number of transaction records saved in the first (default) transaction file.

Syntax *A%* = TRANSACTION_COUNT

Remarks "*A%*" is an integer variable to be assigned to the result.

Example

```
...
DataCount:
  DataCount% = TRANSACTION_COUNT
  CLS
  PRINT DataCount%, "Transaction data is saved."
  RETURN
...
```

See Also TRANSACTION_COUNT_EX

TRANSACTION_COUNT_EX

Purpose To get the total number of transaction records saved in a specified transaction file.

Syntax *A%* = TRANSACTION_COUNT_EX(*file%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. The command TRANSACTION_COUNT_EX(1) works the same as the command TRANSACTION_COUNT.

Example

```
...
DataCount_1:
   DataCount% = TRANSACTION_COUNT_EX(1)
   CLS
   PRINT DataCount%, "Data in transaction file 1."
   RETURN
   ...
```

See Also TRANSACTION_COUNT

UPDATE_TRANSACTION

Purpose To update a transaction record in the first (default) transaction file.

Syntax UPDATE_TRANSACTION(*N%, data$*)

Remarks "*N%*" is an integer variable, indicating the ordinal number of the transaction record to be updated.

"*data$*" is a string variable, representing the character string to replace the old data.

Example

```
...
UpdateTransaction:
   UPDATE_TRANSACTION(Num%, NewData$)
   RETURN
   ...
```

See Also GET_TRANSACTION_DATA$, SAVE_TRANSACTION, UPDATE_TRANSACTION_EX

UPDATE_TRANSACTION_EX

Purpose To update a transaction record in a specified transaction file.

Syntax UPDATE_TRANSACTION_EX(*file%, N%, data$*)

Remarks "*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. The command UPDATE_TRANSACTION_EX(1, N%, data$) works the same as the command UPDATE_TRANSACTION(N%, data$).

"*N%*" is an integer variable, indicating the ordinal number of the transaction record to be updated.

"*data$*" is a string variable, representing the character string to replace the old data.

Example

```
...
UpdateTransaction_1:
  UPDATE_TRANSACTION_EX(1, Num%, NewData$)
  RETURN
  ...
```

See Also GET_TRANSACTION_DATA_EX$, SAVE_TRANSACTION_EX, UPDATE_TRANSACTION

5.20.2 DBF Files and IDX Files

This one is an index sequential file structure. Table look-up and report generation is easily supported by using index sequential file routines. There are actually two types of files associated with this file structure, namely, *DBF files* and *IDX files*.

A DBF file has a fixed record length structure. This is the file that stores the data records (members), whereas, the associated IDX files are the files that keep the information of the position of each record stored in the DBF file. Yet, such index files are re-arranged (sorted) according to some specific key values. In addition to the IDX files that are explicitly created by the user, the BASIC run-time maintains a default IDX file which keeps the original data sequence.

A library would be a good example to illustrate how DBF and IDX files work. When you are trying to find a specific book in a library, you always start from the index. The book can be found by looking into the index categories of book title, writer, publisher, ISBN number, etc. All these index entries are sorted in ascending order for easy lookup according to some specific information of books (book title, writer, publisher, ISBN number, etc.) When the book is found in the index, it will tell you where the book is actually stored.

As you can see, the books kept in the library are analogous to the data records stored in the DBF file, and, the various index entries are just its associate IDX files. Some information (book title, writer, publisher, ISBN number, etc.) in the data records is used to create the IDX files.

For portable terminals, a BASIC program can have up to 5 DBF files. Each DBF file can have up to 3 associated IDX files, and each of them is identified by its key (index) number. The valid key numbers are from 1 to 3. The length of the record in the DBF file is limited to 255 bytes.

Data records are not manipulated directly from the DBF file but rather through its associated IDX files. The value of file pointers of the IDX files (index pointers) does not represent the address of the data records stored in the DBF file. It indicates the sequence number of a specific data record in the IDX file.

ADD_RECORD

Purpose To add a record to a specified DBF file.

Syntax ADD_RECORD(*file%, data$*)

Remarks "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*data$*" is a string variable, representing the character string which the user intends to add to the specified DBF file.

Example

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$, 2, 1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-2)
  IF CmdIdentifier$ = "+" THEN
    ADD_RECORD(DBFNum%, CardID$)
  ELSE
  ...
```

See Also DEL_RECORD

DEL_RECORD

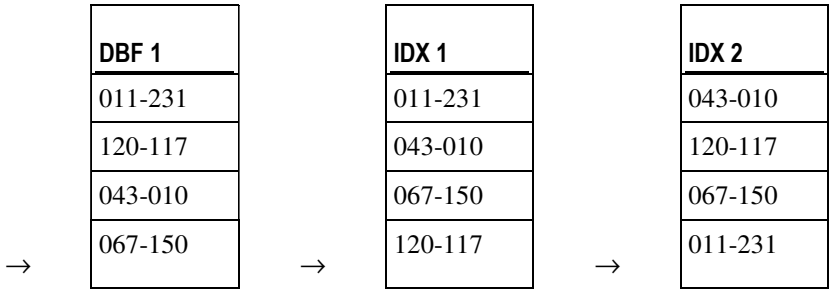
Purpose To delete the record pointed by the file pointer in a specified DBF file.

Syntax DEL_RECORD(*file%* [,*index%*])

Remarks "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

For example, if DBF 1 contains four records: 011-231, 120-117, 043-010, 067-150. The key (index) of the first associate IDX is defined as starting at position 1 with length of 3, and the key (index) of the second associate IDX is defined as starting at position 5 with length of 3. All the file pointers of the DBF and IDX files are currently pointing to the last record.



Then, DEL_RECORD(1) will delete 067-150, DEL_RECORD(1,1) will delete 120-117, DEL_RECORD(1,2) will delete 011-231.

Example

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)

  DBFNum% = VAL(MID$(Cmd$, 2, 1))
  IDXNum% = VAL(MID$(Cmd$, 3, 1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
  IF CmdIdentifier$ = "-" THEN
    DEL_RECORD(DBFNum%, IDXNum%)
  ELSE
  ...
```

See Also ADD_RECORD, EMPTY_FILE

EMPTY_FILE

Purpose To remove all the records from a specified DBF file.

Syntax EMPTY_FILE(*file%*)

Remarks "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

Example

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$, 2, 1))
  IDXNum% = VAL(MID$(Cmd$, 3, 1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
  IF CmdIdentifier$ = "!" THEN
    EMPTY_FILE(DBFNum%)
  ELSE
  ...
```

See Also DEL_RECORD

FIND_RECORD

Purpose To search for records in a specified DBF file that matches the key string with respect to a specified IDX.

Syntax *A%* = FIND_RECORD(*file%, index%, key$*)

Remarks "*A%*" is an integer variable to be assigned to the result.

"*file%%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed.

"*key$*" is a string variable, representing the character string which indicates the matching string to be found.

If any record member in the DBF file matches the key string with respect to the IDX file, FIND_RECORD will return 1, and the file pointer of the IDX file will point to the first record with the matching string.

If there is no match, the file pointer will point to the first record whose index value is greater than the vale of "*key$*".

Example

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$, 2, 1))
```

```
IDXNum% = VAL(MID$(Cmd$, 3, 1))
CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
IF CmdIdentifier$ = "?" THEN
   IF FIND_RECORD(DBFNum%, IDXNum%, CardID$) = 1 THEN
      PRINT "Data is found in DBF.", DBFNum%
   ELSE
      PRINT "Data is not found in DBF.", DBFNum%
   END IF
ELSE
   ...
```

GET_RECORD$

Purpose To get a record in a specified DBF file, which the file pointer of a specified IDX file is pointing to.

Syntax *A$* = GET_RECORD(*file%* [,*index%*])

Remarks "*A$*" is a string variable to be assigned to the result.

"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

Example

```
ON COM(1) GOSUB BcrData_1
...
BcrData_1:
   BEEP(2000,5)
   ID$ = GET_READER_DATA$(1)
   IF FIND_RECORD(DBFNum%, IDXNum%, ID$) = 1 THEN
      Data$ = GET_RECORD$(DBFNum%, IDXNum%)
      Item$ = MID$(Data$, LEN(Data$)-IDLeng%-ItemLeng%)
      Note$ = RIGHT$(Data$, LEN(Data$)-IDLeng%-ItemLeng%)
      LOCATE 1,1
      PRINT "ID     :", Data$
      LOCATE 2,1
      PRINT "Item :", Item$
      LOCATE 3,1
      PRINT "Note :", Note$
```

```
ELSE
…
```

See Also UPDATE_RECORD

GET_RECORD_NUMBER

Purpose To get the ordinal number of the record pointed to by the file pointer of a specified DBF file and IDX file.

Syntax *A%* = GET_RECORD_NUMBER(*file%* [,*index%*])

Remarks "*A%*" is an integer variable to be assigned to the number.

"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

Example A% = GET_RECORD_NUMBER(1,1)

MOVE_TO

Purpose To move the file pointer, of a specified DBF and IDX files, to a specified position.

Syntax MOVE_TO(*file%* [,*index%*], *record_number%*)

Remarks "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

"*record_number%*" is a positive integer variable, indicating the ordinal number of the record where the file pointer is moved to.

Example MOVE_TO(1,1,20)

See Also MOVE_TO_NEXT, MOVE_TO_PREVIOUS

MOVE_TO_NEXT

Purpose To move the file pointer, of a specified DBF and IDX files, one record forward.

Syntax MOVE_TO_NEXT(*file%* [,*index%*])

Remarks "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

Example MOVE_TO_NEXT(1,1)

See Also MOVE_TO, MOVE_TO_PREVIOUS

MOVE_TO_PREVIOUS

Purpose To move the file pointer, of a specified DBF and IDX files, one record backward.

Syntax MOVE_TO_PREVIOUS(*file%* [,*index%*])

Remarks "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

Example MOVE_TO_PREVIOUS(1,1)

See Also MOVE_TO, MOVE_TO_NEXT

RECORD_COUNT

Purpose To get the total number of the records in a specified DBF file.

Syntax *A%* = RECORD_COUNT(*file%*)

Remarks "*A%*" is an integer variable to be assigned to the result.

"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

Example TotalRecord_1% = RECORD_COUNT(1)

UPDATE_RECORD

Purpose To update the record, which the file pointer of a specified DBF and IDX files is pointing to.

Syntax UPDATE_RECORD(*file%, index%, data$*)

Remarks "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

"*data$*" is a string variable, representing the character string to replace the old data.

Example

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$, 2, 1))
  IDXNum% = VAL(MID$(Cmd$, 3, 1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
  IF CmdIdentifier$ = "&" THEN
    UPDATE_RECORD(DBFNum%, IDXNum%, CardID$)
  ELSE
  ...
```

See Also GET_RECORD$

5.20.3 Error Code

The command GET_FILE_ERROR returns the error code, which is a number that indicates the result of the last file manipulation. A value other than 0 indicates error.

GET_FILE_ERROR

Purpose To get the error code of the previous file manipulation command.

Syntax *A%* = GET_FILE_ERROR

Remarks "*A%*" is an integer variable to be assigned to the result.

If there is no error, it returns 0.

If it returns a value other than 0, possible error code and its interpretation will be listed as follows.

Error Code	Interpretation
10	No free memory for file extension.

For other types of error, i.e. invalid file ID, it will cause a run-time error.

Example

```
...
ADD_RECORD(1,Data$)
IF (GET_FILE_ERROR = 10) THEN
   ErrorMessage$ = "No free file space."
END IF
...
```

5.21 Memory

This section describes the commands relating to the flash memory, SRAM, and the Smart Media Card (SMC).

FLASH_READ$

Purpose To read a data string from the flash memory.

Syntax *A$* = FLASH_READ$(*N%*)

Remarks "*A$*" is a string variable to be assigned to the result.

"*N%*" is an integer variable in the range of 1 to 256, indicating the ordinal number of the record.

Example

```
A$ = FLASH_READ$(3)                    ' read the 3rd record
```

See Also FLASH_WRITE

FLASH_WRITE

Purpose To write a data string to the flash memory. Up to 256 records can be saved to the flash memory.

Syntax *A%* = FLASH_WRITE(*N%, A$*)

Remarks "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
1	Write flash memory successfully.
-1	The BASIC program is too large; no free flash memory available.
-2	Error command for erasing the flash memory.
-3	The given index is out of the range.
-4	Fail to write (probably flash memory is not erased yet or something goes wrong).

"*N%*" is an integer variable in the range of 1 to 256, indicating the ordinal number of the record.

"*A$*" is a string variable, representing the data string to be saved to the flash memory.

Before writing any data to the flash memory, it is necessary to use the following command to erase the flash memory first:

err% = FLASH_WRITE(0,"ERASE")

Note that the index must be 0, and the string must be "ERASE". After erasing the flash memory, you can then write data to it by one record at a time. However, when you need to write data to any used index number, the whole flash memory needs to be erased again; otherwise, the write command will fail.

Example err% = FLASH_WRITE(1, "data number#1)

...

err% = FLASH_WRITE(256, "data number#256)

See Also FLASH_READ$

FREE_MEMORY

Purpose To get the size of free data memory (SRAM) in bytes.

Syntax *A&* = FREE_MEMORY

Remarks "*A&*" is a long integer variable to be assigned to the result.

Example PRINT "Free memory = ", FREE_MEMORY

See Also RAM_SIZE, ROM_SIZE

RAM_SIZE

Purpose To get the size of data memory (SRAM) in kilo-bytes.

Syntax *A%* = RAM_SIZE

Remarks "*A%*" is an integer variable to be assigned to the result.

Example PRINT "SRAM size = ", RAM_SIZE

See Also FREE_MEMORY, ROM_SIZE

ROM_SIZE

Purpose To get the size of program memory (Flash) in kilo-bytes.

Syntax *A%* = ROM_SIZE

Remarks "*A%*" is an integer variable to be assigned to the result.

Example PRINT "Flash size = ", ROM_SIZE

See Also FREE_MEMORY, RAM_SIZE

5.22 Debugging Commands

The command *START_DEBUG* will write the activities happening on the system to a specified COM port. It is very useful when the user needs to monitor the system or diagnose the problem.

When *START_DEBUG* is executed, the system will send a series of messages to a specified COM port until the command *STOP_DEBUG* is executed. The following is a list of the debug messages received when executing a sample BASIC program.

Note: Please refer to Appendix III for a description of the debug messages.

```
*   L(7), T(0)
    ADD_RECORD(1,"10001 Justin Jan
    0830093011301300113015001800 2000")
```

```
*   L(42), T(0)
    ON_NET(316)
```

```
*   L(8), T(0)
*   L(9), T(0)
    ASGN(2)
*   L(10), T(0)
    ASGN(3)
*   L(11), T(0)
    ASGN("Terminal 510")
*   L(12), T(0)
    ASGN("510AC_100.BAS")
*   L(13), T(0)
    ...
*   L(25), T(0)
    ARY(1)
    ASGN("OKGood Morning!    ")
    ...
*   L(39), T(0)
    SET_COM(1,1,1,2,1)
*   L(40), T(0)
    OPEN_COM(1)
    ...
*   L(41), T(0)
    START_NETWORK
```

```
*   L(43), T(0)
    ON_ENQUIRY(128)
    ...
*   GOTO(68)
    L(68), T(0)
*   L(69), T(0)
*   L(70), T(0)
    GOTO(68)
    ...
*   L(69), T(0)
    EVENT(16)
*   L(79), T(1)
*   L(80), T(1)
    OFF_READER(1)
*   L(81), T(1)
    OFF_READER(2)
*   L(82), T(1)
    CLS
*   L(83), T(1)
    HIDE_CALENDAR
*   L(84), T(1)
    BEEP(...)
```

START_DEBUG

Purpose To start the debug function.

Syntax START_DEBUG(*N%, Baudrate%, Parity%, Data%, Handshake%*)

Remarks

Parameters	Values	Remarks
N%	1 or 2	Indicates which COM port is to be set.
Baudrate%	1: 115200 bps 2: 76800 bps 3: 57600 bps 4: 38400 bps 5: 19200 bps 6: 9600 bps 7: 4800 bps 8: 2400 bps	Specifies the baud rate of the COM port.
Parity%	1: None 2: Odd 3: Even	Specifies the parity of the COM port.
Data%	1: 7 data bits 2: 8 data bits	Specifies the data bits of the COM port.
Handshake%	1: None 2: CTS/RTS 3: XON/XOFF	Specifies the method of flow control for the COM port.

If a certain COM port has been used in the BASIC program, it is better to use another COM port for debugging to avoid conflicts.

Example

```
START_DEBUG (1,1,1,2,1)        ' use COM1 to send debug messages
                               ' the COM port properties are 115200,
                                 None, 8, No handshake
```

See Also STOP_DEBUG

STOP_DEBUG

Purpose To terminate the debug function.

Syntax STOP_DEBUG

Remarks This is the counter command of START_DEBUG.

Example STOP_DEBUG

See Also START_DEBUG

5.23 Reserved Host Commands

There are some commands reserved for the host computer to read/remove data of the transaction file, or to adjust the system time. The user's BASIC program does not need to do any processing because these tasks will be processed by the background routines of the BASIC run-time.

Note: Each reserved command is ended with a carriage return, which can be changed by the *COM_DELIMITER* function. If any format error occurs, the terminal would return "NAK".

CLEAR

Purpose To erase data of a specified transaction file.

Syntax *A$* = CLEAR

A$ = CLEAR(*file%*)

Remarks "*A$*" is a string variable to be assigned to the result.

Value	Meaning
OK	The command is processed successfully.
NAK	Any format error occurs.

The CLEAR command will clear data of the first transaction file, which is the default one.

"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file is to be erased.

Example CLEAR3 ' to delete data of the 3rd transaction file

See Also REMOVE

READ

Purpose To read the top most record of a specified transaction file.

Syntax *A$* = READ

A$ = READ(*file%*)

Remarks "*A$*" is a string variable to be assigned to the result; it may be the desired data string if the command is successfully processed.

Otherwise, it may have one of the values as follows:

Value	Meaning
OVER	There is no data in the transaction file.
NAK	Any format error occurs.

The READ command will read the top most record of the first transaction file, which is the default one.

"*N%*" is an integer variable in the range of 1 to 6, indicating of which transaction file the record is to be read.

Example
```
READ1                    ' to read a record from the first
                         transaction file
```

REMOVE

Purpose To delete one record from the top of a specified transaction file.

Syntax *A$* = REMOVE

A$ = REMOVE(*file%*)

Remarks "*A$*" is a string variable to be assigned to the result.

Value	Meaning
NEXT	The command is processed successfully.
OVER	There is no more data.
NAK	Any format error occurs.

The REMOVE command will delete one record from the top of the first transaction file, which is the default one.

"*N%*" is an integer variable in the range of 1 to 6, indicating of which transaction file the record is to be deleted.

Example
```
REMOVE2                  ' to delete a record from the 2nd
                         transaction file
```

See Also CLEAR

TR

Purpose To get the current system time.

Syntax *A$* = TR

Remarks "*A$*" is a string variable to be assigned to the result, which is in the form of "yyyymmddhhnnss" as described below.

yyyy:	4 digits for year
mm:	2 digits for month
dd:	2 digits for day
hh:	2 digits for hour, in 24-hour format
nn:	2 digits for minute
ss:	2 digits for second

Otherwise, it may return one of the values below.

Value	Meaning
NAK	Any format error occurs.

Example TR

See Also TW

TW

Purpose To set new system time.

Syntax *A$* = TWyyyymmddhhnnss

Remarks "*A$*" is a string variable to be assigned to the result.

Value	Meaning
OK	The command is processed successfully.
NAK	Any format error occurs.

yyyy:	4 digits for year
mm:	2 digits for month
dd:	2 digits for day
hh:	2 digits for hour, in 24-hour format
nn:	2 digits for minute
ss:	2 digits for second

Example TW20050520103000 ' set system time as 2005/May 20/10:30:00

See Also TR

APPENDIX I

Barcode Setting

The data terminals support the decodability of several barcode symbologies, such as:

- Code 39
- Italy Pharmacode
- French Pharmacode
- Industrial 2 of 5
- Interleave 2 of 5
- Matrix 2 of 5
- Codabar
- Code 93
- Code 128
- UPCE
- UPCE with Addon2
- UPCE with Addon5
- EAN8
- EAN8 with Addon2
- EAN8 with Addon5
- EAN13 & UPCA
- EAN13 with Addon2
- EAN13 with Addon5
- MSI
- Plessey
- Telepen
- RSS14
- RSS14 Limited
- RSS14 Expanded

The BASIC Compiler provides the user a menu-driven interface to configure the decodability of these barcode symbologies, as well as the scanner behavior. The "Barcode Setting" window lets the user check the box in front of a barcode type to enable the decodability of this barcode symbology. For some of the supported barcode symbologies, the user may click the "Configure" button to do more configurations, such as enable/disable the checksum verification.

This appendix describes associated symbology parameters and scanner parameters.

Symbology Parameters

Code 39

Transmit Start / Stop	This parameter specifies whether the start/stop characters of Code 39 are included in the data being transmitted.
Verify Checksum	If this parameter is enabled, the target terminal will perform checksum verification when decoding Code 39 barcodes. If the checksum is incorrect, the barcode will not be accepted.
Transmit Checksum	If this parameter is enabled, the checksum character will be included in the data being transmitted.
Code 39 Full ASCII	User can check the box to read Full ASCII Code 39.

Italian/French Pharmacode

Transmit Checksum	If this parameter is enabled, the checksum character will be included in the data being transmitted.

Industrial/Interleave/Matrix 2 of 5

Start / Stop Selection	This parameter provides the readability of all 2 of 5 symbology variants. For example, flight tickets actually use an Industrial 2 of 5 barcode but with Interleave 2 of 5 start/stop. In order to read this barcode, the start/stop selection parameter of Industrial 2 of 5 should set to "Interleave 25".
Verify Checksum	If this parameter is enabled, the target terminal will perform checksum verification when decoding these barcodes. If the checksum is incorrect, the barcode will not be read.
Transmit Checksum	If this parameter is enabled, the checksum character will be included in the data being transmitted.

Length Qualification

Because of the weak structure of the 2 of 5 barcodes, it is possible to make a "short scan" error. To prevent the "short scan" error, user can define the "Length Qualification" settings to insure that the correct code is read by qualifying the allowable code length. The barcode can be qualified by "Fixed Length" or "Max/Min Length". If "Fixed Length" is selected, up to 2 fixed lengths can be specified. If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. The target terminal will only accept those barcodes with lengths that fall between max/min lengths specified.

Codabar

Start / Stop Character

User can select no start/stop characters or one of the four different start/stop character pairs, i.e., abcd/abcd, abcd/tn*e, ABCD/ABCD, and ABCD/TN*E, to be included in the data being transmitted.

UPCE

Convert to UPCA

If this parameter is enabled, the read UPCE barcode will be expanded into UPCA, and the next processing will follow the parameters configured for UPCA.

Transmit System Number

If this parameter is enabled, the system number will be included in the data being transmitted.

Transmit Checksum

If this parameter is enabled, the checksum character will be included in the data being transmitted.

EAN8

Convert to EAN13

If this parameter is enabled, the EAN 8 read will be expanded into EAN 13, and the next processing will follow the parameters configured for EAN 13.

Transmit Checksum

If this parameter is enabled, the checksum character will be included in the data being transmitted.

EAN13 & UPCA

ISBN Conversion	If this parameter is enabled, the EAN 13 codes starting with 978 and 979 will be converted to ISBN code.
ISSN Conversion	If this parameter is enabled, the EAN 13 codes starting with 977 will be converted to ISSN code.
Transmit Checksum	If this parameter is enabled, the EAN 13 checksum character will be included in the data being transmitted.
Transmit UPCA System Number	If this parameter is enabled, the UPCA system number will be included in the data being transmitted.
Transmit UPCA Checksum	If this parameter is enabled, the UPCA checksum character will be included in the data being transmitted.

MSI

Checksum Verification	User can select one of the three kinds of checksum calculations, i.e., Single Modulo 10, Double Modulo 10, and Modulo 11 & 10, to verify MSI code. If the checksum character is incorrect, the barcode will not be read.
Checksum Transmission	This parameter specifies how the checksum is to be transmitted. User can select from “Last digit not transmitted”, “Transmitted”, and “Last 2 digits not transmitted”.
Length Qualification	Because of the weak structure of the MSI code, it is possible to make a "short scan" error. To prevent the "short scan" error, user can define the "Length Qualification" settings to insure that the correct code is read by qualifying the allowable code length. The barcode can be qualified by "Fixed Length" or "Max/Min Length". If "Fixed Length" is selected, up to 2 fixed lengths can be specified. If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. The target terminal will only accept MSI code with lengths that fall between max/min lengths specified.

Plessey

Convert to UK Plessey	If this parameter is enabled, the target terminal will change each occurrence of the character 'A' to character 'X' in the code.

Checksum Transmission If this parameter is enabled, the checksum characters (two characters) will be transmitted together with data.

Telepen

Original Telepen (Numeric) If this parameter is enabled, the target terminal supports Telepen in numeric numbers only.

AIM Telepen (Full ASCII) If this parameter is enabled, the target terminal supports Telepen in full ASCII code, including all the alphanumeric and special characters.

RSS14

Transmit Code ID If this parameter is enabled, the default Code ID will be included in the data being transmitted.

Transmit Application ID If this parameter is enabled, the Application ID will be included in the data being transmitted.

Transmit Check Digit If this parameter is enabled, the check digit will be included in the data being transmitted.

RSS14 Limited

Transmit Code ID If this parameter is enabled, the default Code ID will be included in the data being transmitted.

Transmit Application ID If this parameter is enabled, the Application ID will be included in the data being transmitted.

Transmit Check Digit If this parameter is enabled, the check digit will be included in the data being transmitted.

RSS14 Expanded

Transmit Code ID — If this parameter is enabled, the default Code ID will be included in the data being transmitted.

Scanner Parameters

The user can define the scanner parameters for reader ports 1 and 2.

> Note: The settings for reader port 2 on the 7xx series data terminals will be ignored because they support only one reader port.

Scan Mode

Auto Off Mode — This is the default scan mode.

The reader will start scanning once the scan trigger is pressed. The scanning goes on until one of the following events happens.

- a barcode is read
- time out (preset scanning period expires)

Auto Power Off Mode — The reader will start scanning once the scan trigger is pressed. The scanning goes on until one of the following events happens.

- time out (preset scanning period expires)

Unlike the Auto Off mode, the reader will continue scanning and the scanning time-out period is re-counted whenever there is a successful read.

Continuous Mode — The reader is always scanning but will decode the same barcode just once. To read the same barcode multiple times, the barcode must be taken away from the scanning line and then replaced.

Repeat Mode — The reader is always scanning just like in the Continuous Mode. But in this mode the scan trigger acts like a "re-transmit button".

This scan mode is most useful when the same barcode is to be read many times. If it is triggered within 1 second after a good read, the same data will be re-transmitted without actually reading the barcode.

Such re-transmission can be triggered as many times as the user desires, as long as the time interval between each triggering does not exceed 1 second.

Test Mode The reader is always scanning and will decode repeatedly even with the same barcode.

Momentary Mode The reader will be scanning as long as the switch is pressed.

Alternate Mode The reader will start scanning once the switch is triggered. It will continue scanning until the scan trigger is pressed again.

Aiming Mode This mode is used when two barcodes are printed too close to each other.

By selecting this mode, the user needs to trigger twice to decode.

That is, the first trigger is for aiming only, while the second trigger will truly start to decode. After the first trigger, the scanner will keep on scanning for 1 second so that the user may take aim. But the user must press the second trigger within this period (1 second by default); otherwise, it will be reset and the user has to take aim again.

The scanning goes on until one of the following events happens.

- a barcode is read
- time out (preset scanning period expires)

Laser Mode This scan mode is most often used on laser scanners. The reader will start scanning once the scan trigger is pressed.

The scanning goes on until one of the following events happens.

- a barcode is read
- the scan trigger is released
- time out (preset scanning period expires)

Read Redundancy

This parameter is used to specify the level of reading security. The user must compromise between reading security and decoding speed.

No Redundancy If "No Redundancy" is selected, one successfully decoded barcode will make the reading valid and induce the "READER Event".

One times

Two times If "Three Times" is selected, it will take a total of four consecutive successful decodings of the same barcode to make the reading valid.

Three times The higher the reading security is (that is, the more redundancy the user selects), the slower the reading speed gets.

Time-Out

This parameter is used to limit the maximum scanning time interval when the scan mode is set as

- Auto Off mode
- Auto Power Off mode
- Aiming mode
- Laser mode

The time interval is specified in units of second, ranging from 0 to 255. It is set 10 seconds by default.

Negative Barcode

Normally, barcodes are printed with the color of the bars darker than that of the spaces. But for negative barcodes, they are printed in the opposite sense, just like the negative films. The spaces of the negative barcodes are printed with a color darker than that of the bars. User can check the box of “Read Negative Barcode” to enable the readability of negative barcodes.

APPENDIX II

Run-Time Error Table

Error Code	Explanation
1	Unknown operator
2	Operand count mismatch
3	Type mismatch
4	Can't perform type conversion
5	No available temp string
6	Illegal operand
7	Not an L-value
8	Float error
9	Bad array subscript
10	Unknown function
11	Illegal function call
12	Return without GOSUB

APPENDIX III

Debugging Messages

Debugging messages indicate the activities happening on the system. The common debugging messages are listed as follows.

Message	Explanation
ABS(N)	Indicating the command ABS is processed.
ADD(N1%,N2%)	Indicating an addition is processed.
ADD_RECORD(file%,data$)	Indicating the command ADD_RECORD is processed.
ALPHA_LOCK(status%)	Indicating the command ALPHA_LOCK is processed.
AND	Indicating the logical operation AND is processed.
ARY(N%)	Indicating an N-element array is declared.
ASC(X$)	Indicating the command ASC is processed.
ASGN(A)	Indicating that the value A is assigned to the variable. A could be an integer, long integer, character, string, or any type.
AUTO_OFF(N%)	Indicating the command AUTO_OFF is processed. N% is the assigned time interval.
BACK_LIGHT_DURATION(N%)	Indicating the command BACK_LIGHT_DURATION is processed. N% is the assigned time interval.
BACKUP_BATTERY	Indicating the command BACKUP_BATTERY is processed.
BEEP(...)	Indicating the command BEEP is processed.
BT_INQUIRY$	Indicating the command BT_INQUIRY$ is processed.
BT_PAIRING(addr$,type%)	Indicating the command BT_PAIRING is processed.
CHANGE_SPEED(N%)	Indicating the command CHANGE_SPEED is processed. N% is the selection of the speed.
CHECK_RF_BASE	Indicating the command CHECK_RF_BASE is processed.
CHECK_RF_SEND	Indicating the command CHECK_RF_SEND is processed.
CHR$(N%)	Indicating the command CHR is processed.
CIRCLE(...)	Indicating the command CIRCLE is processed.
CLOSE_COM(N%)	Indicating the command CLOSE_COM is processed. N% is the number of the COM port.
CLR_KBD	Indicating the command CLR_KBD is processed.
CLR_RECT(...)	Indicating the command CLR_RECT is processed.
CLS	Indicating the command CLS is processed.

CODE_TYPE	Indicating the command CODE_TYPE is processed.
COM_DELIMITER(N%,C%)	Indicating the command COM_DELIMITER is processed.
CURSORX	Indicating the command CURSOR_X is processed.
CURSORY	Indicating the command CURSOR_Y is processed.
DATE$	Indicating the system date is inquired.
DATE$(X$)	Indicating the system date is updated. X$ is the new system date.
DAY_OF_WEEK	Indicating the command DAY_OF_WEEK is processed.
DEL_RECORD(file%[,index%])	Indicating the command DEL_RECORD is processed.
DEL_TRANSACTION_DATA(N%)	Indicating the command DEL_TRANSACTION_DATA is processed. N% is the number of records to be deleted.
DEL_TRANSACTION_DATA_EX(file%,N%)	Indicating the command DEL_TRANSACTION_DATA_EX is processed.
DISABLE_READER(N%)	Indicating the command DISABLE READER is processed. N% is the number of the reader port.
DISABLE_TOUCHSCREEN	Indicating the command DISABLE_TOUCHSCREEN is processed.
DIV(N1%,N2%)	Indicating a division is processed.
DNS_RESOLVER(A$)	Indicating the command DNS_RESOLVER is processed.
EMPTY_FILE(file%)	Indicating the command EMPTY_FILE is processed. file% is the number of the DBF file.
EMPTY_TRANSACTION	Indicating the command EMPTY_TRANSACTION is processed.
EMPTY_TRANSACTION_EX(file%)	Indicating the command EMPTY_TRANSACTION_EX is processed. file% is the number of the transaction file.
ENABLE_READER(N%)	Indicating the command ENABLE READER is processed. N% is the number of the reader port.
ENABLE_TOUCHSCREEN	Indicating the command ENABLE TOUCHSCREEN is processed.
EQU? (N1%,N2%)	Indicating the decision "IF N1% = N2%" is processed.
EVENT(0)	Indicating the "COM(1) EVENT" happens.
EVENT(1)	Indicating the "COM(2) EVENT" happens.
EVENT(2)	Indicating the "COM(3) EVENT" happens.
EVENT(3)	Reserved.
EVENT(4)	Reserved.
EVENT(5)	Reserved.
EVENT(6)	Reserved.
EVENT(7)	Reserved.
EVENT(8)	Reserved.

EVENT(9)	Indicating the "TIMER(1) EVENT" happens.
EVENT(10)	Indicating the "TIMER(2) EVENT" happens.
EVENT(11)	Indicating the "TIMER(3) EVENT" happens.
EVENT(12)	Indicating the "TIMER(4) EVENT" happens.
EVENT(13)	Indicating the "TIMER(5) EVENT" happens.
EVENT(14)	Indicating the "ON MINUTE EVENT" happens.
EVENT(15)	Indicating the "ON HOUR EVENT" happens.
EVENT(16)	Indicating the "READER(1) EVENT" happens.
EVENT(17)	Indicating the "READER(2) EVENT" happens.
EVENT(18)	Indicating the "FUNCTION(1) EVENT" happens.
EVENT(19)	Indicating the "FUNCTION(2) EVENT" happens.
EVENT(20)	Indicating the "FUNCTION(3) EVENT" happens.
EVENT(21)	Indicating the "FUNCTION(4) EVENT" happens.
EVENT(22)	Indicating the "FUNCTION(5) EVENT" happens.
EVENT(23)	Indicating the "FUNCTION(6) EVENT" happens.
EVENT(24)	Indicating the "FUNCTION(7) EVENT" happens.
EVENT(25)	Indicating the "FUNCTION(8) EVENT" happens.
EVENT(26)	Indicating the "FUNCTION(9) EVENT" happens.
EVENT(27)	Indicating the "FUNCTION(10) EVENT" happens.
EVENT(28)	Indicating the "FUNCTION(11) EVENT" happens.
EVENT(29)	Indicating the "FUNCTION(12) EVENT" happens.
EVENT(30)	Reserved.
EVENT(31)	Indicating the "ESC EVENT" happens.
EXP(N1%,N2%)	Indicating an exponentiation is processed.
FALSE?(N%)	Indicating the "IF" statement or the "WHILE" statement is processed.
FILL_RECT(...)	Indicating the command FILL_RECT is processed.
FIND_RECORD(...)	Indicating the command FIND_RECORD is processed.
FLASH_READ$(N%)	Indicating the command FLASH_READ$ is processed.
FLASH_WRITE(N%,A$)	Indicating the command FLASH_WRITE is processed.
FREE_MEMORY	Indicating the command FREE_MEMORY is processed.
FUNCTION_TOGGLE(status%)	Indicating the command FUNCTION_TOGGLE is processed.
GE? (N1%,N2%)	Indicating the decision "IF N1% >= N2%" is processed.
GET_ALPHA_LOCK	Indicating the command GET_ALPHA_LOCK is processed.

GET_CTS(N%)	Indicating the command GET_CTS is processed. N% is the number of the COM port.
GET_DEVICE_ID	Indicating the command DEVICE_ID is processed.
GET_FILE_ERROR	Indicating the command GET_FILE_ERROR is processed.
GET_IMAGE	Indicating the command GET_IMAGE is processed.
GET_LANGUAGE	Indicating the command GET_LANGUAGE is processed.
GET_NET_PARAMETER$(index %)	Indicating the command GET_NET_PARAMETER$ is processed.
GET_NET_STATUS(index%)	Indicating the command GET_NET_STATUS is processed.
GET_READER_DATA$(N%)	Indicating the command GET_READER_DATA$ is processed. N% is the number of the reader port.
GET_READER_SETTING(N%)	Indicating the command GET_READER_SETTING is processed. N% is the setting number.
GET_RECORD$(file%[,index%])	Indicating the command GET_RECORD$ is processed.
GET_RECORD_NUMBER(file%[,index%])	Indicating the command GET_READER_NUMBER is processed.
GET_RF_CHANNEL	Indicating the command GET_RF_CHANNEL is processed.
GET_RF_ID	Indicating the command GET_RF_ID is processed.
GET_RF_POWER	Indicating the command GET_RF_POWER is processed.
GET_SCREENITEM	Indicating the command GET_SCREENITEM is processed.
GET_SHIFT_LOCK	Indicating the command GET_SHIFT_LOCK is processed.
GET_TARGET_MACHINE$	Indicating the command GET_TARGET_MACHINE$ is processed.
GET_TCPIP_MESSAGE	Indicating the command GET_TCPIP_MESSAGE is processed.
GET_TRANSACTION_DATA$(N%)	Indicating the command GET_TRANSACTION_DATA is processed. N% is the ordinal number of the record to be read.
GET_TRANSACTION_DATA_E X$(file%,N%)	Indicating the command GET_TRANSACTION_DATA_EX is processed.
GOSUB(N%)	Indicating the program branches to a subroutine. N% is the line number of the first line of the subroutine.
GOTO(N%)	Indicating the program branches to line number N%.
GSM_CHANGE_PIN(old$,new$)	Indicating the command GSM_CHANGE_PIN is processed.
GSM_CHECK_PIN(pin$)	Indicating the command GSM_CHECK_PIN is processed.
GSM_SET_PINLOCK(pin$,mode %)	Indicating the command GSM_SET_PINLOCK is processed.
GT? (N1%,N2%)	Indicating the decision "IF N1% > N2%" is processed.
HEX$(N%)	Indicating the command HEX$ is processed.
ICON_ZONE_PRINT(status%)	Indicating the command ICON_ZONE_PRINT is processed.

INKEY$(A$)	Indicating the command INKEY is processed.
INPUT	Indicating the command INOUT is processed.
INPUT_MODE(mode%)	Indicating the command INPUT_MODE is processed.
INSTR([N%,] X$,Y$)	Indicating the command INSTR is processed.
INT(N%)	Indicating the command INT is processed.
IRDA_STATUS(N%)	Indicating the command IRDA_STATUS is processed. N% is the connection or transmission status.
IRDA_TIMEOUT(N%)	Indicating the command IRDA_TIMEOUT is processed. N% is the assigned time interval.
KEY_CLICK(status%)	Indicating the command KEY_CLICK is processed.
L(N%)	Indicating the line number being executed.
LCASE$(X$)	Indicating the command LCASE$ is processed.
LCD_CONTRAST(N%)	Indicating the command LCD_CONTRAST is processed. N% is the contrast level in the range of 1 ~ 8.
LE? (N1%,N2%)	Indicating the decision "IF N1% <= N2%" is processed.
LED(...)	Indicating the command LED is processed.
LEFT$(X$,N%)	Indicating the command LEFT$ is processed.
LEN(X$)	Indicating the command LEN is processed.
LINE(...)	Indicating the command LINE is processed.
LOCATE(N1%,N2%)	Indicating the command LOCATE is processed.
LOCK	Indicating the command LOCK is processed.
LT? (N1%,N2%)	Indicating the decision "IF N1% < N2%" is processed.
MAIN_BATTERY	Indicating the command MAIN_BATTERY is processed.
MENU(Item$)	Indicating the command MENU is processed.
MID$(X$,N%[,M%])	Indicating the command MID$ is processed.
MOD(N1%,N2%)	Indicating a modulo operation is processed.
MOVE_TO(file%[,index%],record_number%)	Indicating the command MOVE_TO is processed. file% is the number of the DBF file; index% is the number of the IDX file; record_number% is the record number to move to.
MOVE_TO_NEXT(file%[,index%])	Indicating the command MOVE_TO_NEXT is processed.
MOVE_TO_PREVIOUS(file%[,index%])	Indicating the command MOVE_TO_PREVIOUS is processed.
MUL(N1%,N2%)	Indicating a multiplication is processed.
NEG (N1%)	Indicating a negation is processed.
NEQ? (N1%,N2%)	Indicating the decision "IF N1% <> N2%" is processed.

NCLOSE(N%)	Indicating the command NCLOSE is processed. N% is the connection number.
NOT	Indicating the logical operation NOT is processed.
NREAD$(N%)	Indicating the command NREAD$ is processed. N% is the connection number.
NWRITE(N%,A$)	Indicating the command NWRITE is processed.
OCT$(N%)	Indicating the command OCT$ is processed.
OFF_ALL	Indicating the command OFF ALL is processed.
OFF_COM(N%)	Indicating the command OFF COM is processed. N% is the number of the COM port.
OFF_ESC	Indicating the command OFF ESC is processed.
OFF_HOUR_SHARP	Indicating the command OFF HOUR_SHARP is processed.
OFF_KEY(number%)	Indicating the command OFF KEY is processed.
OFF_MINUTE_SHARP	Indicating the command OFF MINUTE_SHARP is processed.
OFF_READER(N%)	Indicating the command OFF READER is processed. N% is the number of the reader port.
OFF_TCPIP	Indicating the command OFF TCPIN is processed.
OFF_TIMER(N%)	Indicating the command OFF TIMER is processed. N% is the number of the timer.
OFF_TOUCHSCREEN	Indicating the command OFF TOUCHSCREEN is processed.
ON_COM(N1%,N2%)	Indicating the command ON COM GOSUB is called. N1% is the umber of the COM port; N2% is the line number of the subroutine to branch to.
ON_ESC(N%)	Indicating the command ON ESC GOSUB is called. N% is the line number of the subroutine to branch to.
ON_GOSUB(N%)	Indicating the command ON GOSUB is called. N% is the line number of the subroutine to branch to.
ON_GOTO(N%)	Indicating the command ON GOTO is called. N% is the line number of the subroutine to branch to.
ON_HOUR_SHARP(N%)	Indicating the command ON HOUR_SHARP GOSUB is called. N% is the line number of the subroutine to branch to.
ON_KEY(N%)	Indicating the command ON KEY GOSUB is called. N% is the line number of the subroutine to branch to.
ON_MINUTE_SHARP(N%)	Indicating the command ON MINUTE_SHARP GOSUB is called. N% is the line number of the subroutine to branch to.
ON_READER(N1%,N2%)	Indicating the command ON READER GOSUB is called. N1% is the number of the reader port; N2% is the line number of the subroutine to branch to.
ON_TCPIP(N%)	Indicating the command ON TCPIP GOSUB is called. N% is the line number of the subroutine to branch to.

ON_TIMER(N1%,N2%)	Indicating the command ON TIMER GOSUB is called.
ON_TOUCHSCREEN(N%)	Indicating the command ON TOUCHSCREEN GOSUB is called. N% is the line number of the subroutine to branch to.
OPEN_COM(N%)	Indicating the command OPEN_COM is processed. N% is the number of the COM port.
OR	Indicating the logical operation OR is processed.
POWER_ON(N%)	Indicating the command POWER_ON is processed. N% is the value of the setting.
PRINT(A$)	Indicating the command PRINT is processed.
PUT_PIXEL(...)	Indicating the command PUT_PIXEL is processed.
PUTKEY(N%)	Indicating the command PUTKEY is processed.
RAM_SIZE	Indicating the command RAM_SIZE is processed.
READ_COM$(N%)	Indicating the command READ_COM$ is processed. N% is the number of the COM port.
READER_SETTING(N1%,N2%)	Indicating the command READER_SETTING is processed. N1% is the setting number; N2% is the value of the setting.
RECORD_COUNT(file%)	Indicating the command RECORD_COUNT is processed.
RECTANGLE(...)	Indicating the command RECTANGLE is processed.
RESTART	Indicating the command RESTART is processed.
RETURN(N%)	Indicating the command RETURN is processed. N% is the line number to return, if it is not null.
RIGHT$(X$,N%)	Indicating the command RIGHT$ is processed.
ROM_SIZE	Indicating the command ROM_SIZE is processed.
SAVE_TRANSACTION(data$)	Indicating the command SAVE_TRANSACTION is processed.
SAVE_TRANSACTION_EX(file %,data$)	Indicating the command SAVE_TRANSACTION_EX is processed.
SEARCH_RF_CHANNEL(N%)	Indicating the command SEARCH_RF_CHANNEL is processed. N% is the assigned time interval.
SELECT_FONT(font%)	Indicating the command SELECT_FONT is processed.
SEND_WEDGE(DataString$)	Indicating the command SEND_WEDGE is processed.
SET_COM(...)	Indicating the command SET_COM is processed.
SET_COMM_TYPE(N%,type%)	Indicating the command SET_COMM_TYPE is processed.
SET_CURSOR(status%)	Indicating the command CURSOR is processed.
SET_LANGUAGE(N%)	Indicating the command SET_LANGUAGE is processed. N% is the setting of language.
SET_NET_PARAMETER(index% ,A$)	Indicating the command SET_NET_PARAMETER is processed.

SET_PRECISION(N%)	Indicating the command SET_PRECISION is processed. N% is the numeric precision.
SET_RF_CHANNEL(N%)	Indicating the command SET_RF_CHANNEL is processed. N% is the channel.
SET_RF_ID(N%)	Indicating the command SET_RF_ID is processed. N% is the ID.
SET_RF_POWER(N%)	Indicating the command SET_RF_POWER is processed. N% is the power level.
SET_RF_TIMEOUT(N%)	Indicating the command SET_RF_TIMEOUT is processed. N% is the assigned time interval.
SET_RFID_READ(...)	Indicating the command SET_RFID_READ is processed.
SET_RFID_WRITE(...)	Indicating the command SET_RFID_WRITE is processed.
SET_RTS(N1%,N2%)	Indicating the command SET_RTS is processed. N1% is the number of the COM port; N2% is the RTS status.
SET_SCREENITEMS(...)	Indicating the command SET_SCREENITEMS is processed.
SET_SIGNAREA(...)	Indicating the command SET_SIGNAREA is processed.
SET_VIDEO_MODE(mode%)	Indicating the command SET_VIDEO_MODE is processed.
SET_WEDGE(WedgeSetting$)	Indicating the command SET_WEDGE is processed.
SHIFT_LOCK(N%)	Indicating the command SHIFT_LOCK is processed. N% is the shift lock status.
SHOW_IMAGE(...)	Indicating the command SHOW_IMAGE is processed.
SIGN(N%)	Indicating the command SGN is processed.
SMC_FREE_MEMORY	Indicating the command SMC_FREE_MEMORY is processed.
SMC_SIZE	Indicating the command SMC_SIZE is processed.
SOCKET_CAN_SEND(...)	Indicating the command SOCKET_CAN_SEND is processed.
SOCKET_HAS_DATA(N%)	Indicating the command SOCKET_HAS_DATA is processed. N% is the connection number.
SOCKET_OPEN(N%)	Indicating the command SOCKET_OPEN is processed. N% is the connection number.
START TCPIP	Indicating the command START TCPIP is processed.
STOP_BEEP	Indicating the command STOP BEEP is processed.
STOP TCPIP	Indicating the command STOP TCPIP is processed.
STR$(N%)	Indicating the command STR$ is processed.
STRING$(...)	Indicating the command STRING$ is processed.
SUB(N1%,N2%)	Indicating a subtraction is processed.
SYSTEM_PASSWORD(A$)	Indicating the command SYSTEM_PASSWORD is processed. A$ is the character string to be written as the password.

T(N%)	Indicating the stack's level. When the program branches to a subroutine, the stack's level increases 1; when the program returns, the stack's level decreases 1. It can be used to check if the "stack overflow" problem happens.
TCP_ERR_CODE	Indicating the command TCP_ERR_CODE is processed.
TCP_OPEN(...)	Indicating the command TCP_OPEN is processed.
TIME$	Indicating the system time is inquired.
TIME$(X$)	Indicating the system time is updated. X$ is the new system time.
TIMER	Indicating the command TIMER is processed.
TRANSACTION_COUNT	Indicating the command TRANSACTION_COUNT is processed.
TRANSACTION_COUNT_EX(file%)	Indicating the command TRANSACTION_COUNT_EX is processed.
TRIM_LEFT$(X$)	Indicating the command TRIM_LEFT$ is processed.
TRIM_RIGHT$(X$)	Indicating the command TRIM_RIGHT$ is processed.
UCASE$(X$)	Indicating the command UCASE$ is processed.
UNLOCK	Indicating the command UNLOCK is processed.
UPDATE_RECORD(...)	Indicating the command UPDATE_RECORD is processed.
UPDATE_TRANSACTION(N%,data$)	Indicating the command UPDATE_TRANSACTION is processed.
UPDATE_TRANSACTION_EX(...)	Indicating the command UPDATE_TRANSACTION_EX is processed.
VAL(X$)	Indicating the command VAL is processed.
VALF(X$)	Indicating the command VALR is processed.
VERSION(A$)	Indicating the command VERSION is processed. A$ is the character string to be written as the version information.
VIBRATOR(mode%)	Indicating the command VIBRATOR is processed.
WAIT(duration%)	Indicating the command WAIT is processed.
WAIT_HOURGLASS(...)	Indicating the command WAIT_HOURGLASS is processed.
WEDGE_READY	Indicating the command WEDGE_READY is processed.
WRITE_COM(N%,A$)	Indicating the command WRITE_COM is processed.
XOR	Indicating the logical operation XOR is processed.

Index

@

@AT • 119
@BC • 119
@BS • 119
@CH • 120
@DT • 120
@ER • 120
@GP • 121
@HD • 121
@ID • 121
@ME • 122
@PW • 122
@QT • 123
@SP • 123
@TM • 123
@TO • 124
@UP • 124

A

About Barcode Type • 58, 184
About Bluetooth • 142
About Category of Commands • 23
About COM Ports • 102
About DAT Files • 159
About DBF/IDX Files • 165
About Error Message • 137, 173, 191
About Fonts • 3, 98
About Graphics • 95
About GSM/GPRS • 151
About IEEE 802.11b • 131
About Narrow Band RF • 110
About Network Configuration • 125
About Port Mapping • 102
About RF Models • 110
About RFID • 67
About Runtime Engine • 3, 6, 7
About Scanner • 58, 188
About Spread Spectrum RF • 110
About TCP/IP Networking • 132
About Touch Screen • 98
About Wireless Models • 125
ABS • 25
ADD_RECORD • 166
ALPHA_LOCK • 82
ASC • 37
AUTO_OFF • 52

B

BACK_LIGHT_DURATION • 87
BACKUP_BATTERY • 101
BEEP • 76
BT_INQUIRY$ • 143
BT_PAIRING • 143

C

CHANGE_SPEED • 53
CHECK_RF_BASE • 113
CHECK_RF_SEND • 113
CHR$ • 37
CIRCLE • 96

CLEAR • 179
CLOSE_COM • 103
CLR_KBD • 83
CLR_RECT • 87
CLS • 88
CODE_TYPE • 58
COM_DELIMITER • 106
Commands
  Barcode Reader • 58
  Battery • 101
  Buzzer • 76
  Calendar&Time • 78
  COM Ports • 102
  Debugging • 176
  Decision Structure • 28
  Event Trapping • 41
  File Manipulation • 158
  General • 25
  Host, Reserved • 179
  Keyboard Wedge • 69
  Keypad • 82
  LCD • 87
  LED • 81
  Looping Structures • 31
  Memory • 173
  RF (433 MHz/2.4 GHz) • 109
  RFID Reader • 67
  String Processing • 33
  System • 52
  Vibrator • 77
  Wireless • 125
CURSOR • 88
CURSOR_X • 88
CURSOR_Y • 89

D

DATE$ • 78
DAY_OF_WEEK • 79
DEL_RECORD • 166
DEL_TRANSACTION_DATA • 159
DEL_TRANSACTION_DATA_EX • 159
DEVICE_ID$ • 53
DIM • 25
DISABLE READER • 59
DISABLE_TOUCHSCREEN • 99
DNS_RESOLVER • 132

E

EMPTY_FILE • 167
EMPTY_TRANSACTION • 160
EMPTY_TRANSACTION_EX • 161
ENABLE READER • 60
ENABLE_TOUCHSCREEN • 99
EXIT • 31

F

FILL_RECT • 90
FIND_RECORD • 168
FLASH_READ$ • 173
FLASH_WRITE • 174
FOR ... NEXT • 32
FREE_MEMORY • 175
Frequent Device List • 142, 145
FUNCTION_TOGGLE • 83

G

GET_ALPHA_LOCK • 84
GET_CTS • 106
GET_FILE_ERROR • 173
GET_IMAGE • 90
GET_LANGUAGE • 53
GET_NET_PARAMETER$ • 127
GET_NET_STATUS • 130
GET_READER_DATA$ • 60
GET_READER_SETTING • 61
GET_RECORD$ • 169
GET_RECORD_NUMBER • 170
GET_RF_CHANNEL • 113
GET_RF_ID • 114
GET_RF_POWER • 114
GET_SCREENITEM • 99
GET_SHIFT_LOCK • 84
GET_TARGET_MACHINE$ • 55
GET_TCPIP_MESSAGE • 132
GET_TRANSACTION_DATA$ • 161
GET_TRANSACTION_DATA_EX$ • 161
GET_WLAN_STATUS • 130
GOSUB • 25
GOTO • 26
GSM, Unlock • 157
GSM_CHANGE_PIN • 151
GSM_CHECK_PIN • 151
GSM_SET_PINLOCK • 152

H

HEX$ • 38

I

ICON_ZONE_PRINT • 90

IF ... THEN ... [ELSE...] • 28
IF ... THEN ... {ELSE IF...} [ELSE...] END IF • 28
IF … THEN … END IF • 29
INKEY$ • 84
INPUT • 85
INPUT_MODE • 85
INSTR • 34
INT • 26
IP_CFG or IP_CONFIGURE • 133
IRDA_STATUS • 107
IRDA_TIMEOUT • 108

K

KEY_CLICK • 77

L

LCASE$ • 38
LCD_CONTRAST • 91
LED • 81
LEFT$ • 35
LEN • 34
LINE • 96
LOCATE • 91
LOCK • 51

M

MAIN_BATTERY • 102
MENU • 55
MID$ • 35
MOVE_TO • 170
MOVE_TO_NEXT • 171
MOVE_TO_PREVIOUS • 171

N

NCLOSE • 134
NREAD$ • 134
NWRITE • 134

O

OCT$ • 38
OFF ALL • 42
OFF COM • 42
OFF ESC • 43
OFF HOUR_SHARP • 43
OFF KEY • 44
OFF MINUTE_SHARP • 44
OFF READER • 45
OFF TCPIP • 45
OFF TIMER • 45
OFF TOUCHSCREEN • 46
ON ... GOSUB ... • 29
ON ... GOTO .. • 30
ON COM... GOSUB... • 46
ON ESC GOSUB... • 47
ON HOUR_SHARP GOSUB... • 47
ON KEY... GOSUB... • 48
ON MINUTE_SHARP GOSUB... • 48
ON READER... GOSUB... • 49
ON TCPIP GOSUB... • 49
ON TIMER... GOSUB... • 50
ON TOUCHSCREEN GOSUB... • 50
OPEN_COM • 103

P

POWER_ON • 56
PRINT • 92
PUT_PIXEL • 96
PUTKEY • 86

R

RAM_SIZE • 175
READ • 179
READ_COM$ • 104
READER_SETTING • 61
RECORD_COUNT • 172
RECTANGLE • 97
REM • 26
REMOVE • 180
RESTART • 56
RIGHT$ • 36
ROM_SIZE • 175

S

SAVE_TRANSACTION • 162
SAVE_TRANSACTION_EX • 163
SEARCH_RF_CHANNEL • 115
SELECT_FONT • 98
SEND_WEDGE • 74
SET_COM • 108
SET_COM_TYPE • 104
SET_LANGUAGE • 57
SET_NET_PARAMETER • 127
SET_PRECISION • 27
SET_RF_CHANNEL • 115
SET_RF_ID • 115
SET_RF_POWER • 116
SET_RF_TIMEOUT • 116
SET_RFID_READ • 68

SET_RFID_WRITE • 68
SET_RTS • 109
SET_SCREENITEMS • 100
SET_SIGNAREA • 100
SET_VIDEO_MODE • 97
SET_WEDGE • 75
SGN • 27
SHIFT_LOCK • 86
SHOW_IMAGE • 92
SMC_FREE_MEMORY • 175
SMC_SIZE • 176
SOCKET_CAN_SEND • 134
SOCKET_HAS_DATA • 135
SOCKET_IP$ • 135
SOCKET_OPEN • 136
START TCPIP • 136
START_DEBUG • 178
STOP BEEP • 77
STOP TCPIP • 137
STOP_DEBUG • 178
STR$ • 38
STRING$ • 40
SYSTEM_PASSWORD • 58

T

TCP_ERR_CODE • 137
TCP_OPEN • 138
TIME$ • 79
TIMER • 80
TR • 180
TRANSACTION_COUNT • 163
TRANSACTION_COUNT_EX • 164
TRIM_LEFT$ • 36
TRIM_RIGHT$ • 37
TW • 181

U

UCASE$ • 39
UNLOCK • 52
UPDATE_RECORD • 172
UPDATE_TRANSACTION • 164
UPDATE_TRANSACTION_EX • 164

V

VAL • 39
VALR • 40
VERSION • 58
VIBRATOR • 78

W

WAIT • 81
WAIT_HOURGLASS • 94
WEDGE_READY • 75
WHILE ... WEND • 32
WRITE_COM • 105